Tell us about your PDF experience.

System Namespace

Reference

Contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

Classes

Expand table

AccessViolationException	The exception that is thrown when there is an attempt to read or write protected memory.
Activator	Contains methods to create types of objects locally or remotely, or obtain references to existing remote objects. This class cannot be inherited.
AggregateException	Represents one or more errors that occur during application execution.
AppContext	Provides members for setting and retrieving data about an application's context.
AppDomain	Represents an application domain, which is an isolated environment where applications execute. This class cannot be inherited.
AppDomainSetup	Represents assembly binding information that can be added to an instance of AppDomain.
AppDomainUnloadedException	The exception that is thrown when an attempt is made to access an unloaded application domain.
ApplicationException	Serves as the base class for application-defined exceptions.
ApplicationId	Contains information used to uniquely identify a manifest-based application. This class cannot be inherited.
ArgumentException	The exception that is thrown when one of the arguments provided to a method is not valid.
ArgumentNullException	The exception that is thrown when a null reference (`Nothing` in Visual Basic) is passed to a method that does not accept it as a valid argument.

ArgumentOutOfRangeException	The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
ArithmeticException	The exception that is thrown for errors in an arithmetic, casting, or conversion operation.
Array	Provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime.
ArrayTypeMismatchException	The exception that is thrown when an attempt is made to store an element of the wrong type within an array.
AssemblyLoadEventArgs	Provides data for the AssemblyLoad event.
Attribute	Represents the base class for custom attributes.
AttributeUsageAttribute	Specifies the usage of another attribute class. This class cannot be inherited.
BadImageFormatException	The exception that is thrown when the file image of a dynamic link library (DLL) or an executable program is invalid.
BitConverter	Converts base data types to an array of bytes, and an array of bytes to base data types.
Buffer	Manipulates arrays of primitive types.
CannotUnloadAppDomain Exception	The exception that is thrown when an attempt to unload an application domain fails.
CharEnumerator	Supports iterating over a String object and reading its individual characters. This class cannot be inherited.
CLSCompliantAttribute	Indicates whether a program element is compliant with the Common Language Specification (CLS). This class cannot be inherited.
Console	Represents the standard input, output, and error streams for console applications. This class cannot be inherited.
ConsoleCancelEventArgs	Provides data for the CancelKeyPress event. This class cannot be inherited.
ContextBoundObject	Defines the base class for all context-bound classes.
ContextMarshalException	The exception that is thrown when an attempt to marshal an object across a context boundary fails.

ContextStaticAttribute	Indicates that the value of a static field is unique for a particular context.
Convert	Converts a base data type to another base data type.
DataMisalignedException	The exception that is thrown when a unit of data is read from or written to an address that is not a multiple of the data size. This class cannot be inherited.
DBNull	Represents a nonexistent value. This class cannot be inherited.
Delegate	Represents a delegate, which is a data structure that refers to a static method or to a class instance and an instance method of that class.
DivideByZeroException	The exception that is thrown when there is an attempt to divide an integral or Decimal value by zero.
DllNotFoundException	The exception that is thrown when a DLL specified in a DLL import cannot be found.
DuplicateWaitObjectException	The exception that is thrown when an object appears more than once in an array of synchronization objects.
EntryPointNotFoundException	The exception that is thrown when an attempt to load a class fails due to the absence of an entry method.
Enum	Provides the base class for enumerations.
Environment	Provides information about, and means to manipulate, the current environment and platform. This class cannot be inherited.
EventArgs	Represents the base class for classes that contain event data, and provides a value to use for events that do not include event data.
Exception	Represents errors that occur during application execution.
ExecutionEngineException	The exception that is thrown when there is an internal error in the execution engine of the common language runtime. This class cannot be inherited.
FieldAccessException	The exception that is thrown when there is an invalid attempt to access a private or protected field inside a class.
FileStyleUriParser	A customizable parser based on the File scheme.
FlagsAttribute	Indicates that an enumeration can be treated as a bit field; that is, a set of flags.

FormatException	The exception that is thrown when the format of an argument is invalid, or when a composite format string is not well formed.
FormattableString	Represents a composite format string, along with the arguments to be formatted.
FtpStyleUriParser	A customizable parser based on the File Transfer Protocol (FTP) scheme.
GC	Controls the system garbage collector, a service that automatically reclaims unused memory.
GenericUriParser	A customizable parser for a hierarchical URI.
GopherStyleUriParser	A customizable parser based on the Gopher scheme.
HttpStyleUriParser	A customizable parser based on the HTTP scheme.
IndexOutOfRangeException	The exception that is thrown when an attempt is made to access an element of an array or collection with an index that is outside its bounds.
InsufficientExecutionStack Exception	The exception that is thrown when there is insufficient execution stack available to allow most methods to execute.
InsufficientMemoryException	The exception that is thrown when a check for sufficient available memory fails. This class cannot be inherited.
InvalidCastException	The exception that is thrown for invalid casting or explicit conversion.
InvalidOperationException	The exception that is thrown when a method call is invalid for the object's current state.
InvalidProgramException	The exception that is thrown when a program contains invalid Microsoft intermediate language (MSIL) or metadata. Generally this indicates a bug in the compiler that generated the program. This exception is also thrown when internal runtime implementation limits have been exceeded by the program.
InvalidTimeZoneException	The exception that is thrown when time zone information is invalid.
Lazy<T>	Provides support for lazy initialization.
Lazy<T,TMetadata>	Provides a lazy indirect reference to an object and its associated metadata for use by the Managed Extensibility Framework.

LdapStyleUriParser	A customizable parser based on the Lightweight Directory Access Protocol (LDAP) scheme.
LoaderOptimizationAttribute	Used to set the default loader optimization policy for the main method of an executable application.
LocalDataStoreSlot	Encapsulates a memory slot to store local data. This class cannot be inherited.
MarshalByRefObject	Enables access to objects across application domain boundaries in applications that support remoting.
Math	Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.
MathF	Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.
MemberAccessException	The exception that is thrown when an attempt to access a class member fails.
MemoryExtensions	Provides extension methods for the memory-related and span-related types, such as Memory<T>, ReadOnlyMemory<T>, Span<T>, and ReadOnlySpan<T>.
MethodAccessException	The exception that is thrown when there is an invalid attempt to access a method, such as accessing a private method from partially trusted code.
MissingFieldException	The exception that is thrown when there is an attempt to dynamically access a field that does not exist. If a field in a class library has been removed or renamed, recompile any assemblies that reference that library.
MissingMemberException	The exception that is thrown when there is an attempt to dynamically access a class member that does not exist or that is not declared as public. If a member in a class library has been removed or renamed, recompile any assemblies that reference that library.
MissingMethodException	The exception that is thrown when there is an attempt to dynamically access a method that does not exist.
MTAThreadAttribute	Indicates that the COM threading model for an application is multithreaded apartment (MTA).
MulticastDelegate	Represents a multicast delegate; that is, a delegate that can have more than one element in its invocation list.
MulticastNotSupportedException	The exception that is thrown when there is an attempt to combine two delegates based on the Delegate type instead

	of the MulticastDelegate type. This class cannot be inherited.
NetPipeStyleUriParser	A parser based on the NetPipe scheme for the "Indigo" system.
NetTcpStyleUriParser	A parser based on the NetTcp scheme for the "Indigo" system.
NewsStyleUriParser	A customizable parser based on the news scheme using the Network News Transfer Protocol (NNTP).
NonSerializedAttribute	Indicates that a field of a serializable class should not be serialized. This class cannot be inherited.
NotFiniteNumberException	The exception that is thrown when a floating-point value is positive infinity, negative infinity, or Not-a-Number (NaN).
NotImplementedException	The exception that is thrown when a requested method or operation is not implemented.
NotSupportedException	The exception that is thrown when an invoked method is not supported, or when there is an attempt to read, seek, or write to a stream that does not support the invoked functionality.
Nullable	Supports a value type that can be assigned `null`. This class cannot be inherited.
NullReferenceException	The exception that is thrown when there is an attempt to dereference a null object reference.
Object	Supports all classes in the .NET class hierarchy and provides low-level services to derived classes. This is the ultimate base class of all .NET classes; it is the root of the type hierarchy.
ObjectDisposedException	The exception that is thrown when an operation is performed on a disposed object.
ObsoleteAttribute	Marks the program elements that are no longer in use. This class cannot be inherited.
OperatingSystem	Represents information about an operating system, such as the version and platform identifier. This class cannot be inherited.
OperationCanceledException	The exception that is thrown in a thread upon cancellation of an operation that the thread was executing.
OutOfMemoryException	The exception that is thrown when there is not enough memory to continue the execution of a program.

OverflowException	The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.
ParamArrayAttribute	Indicates that a method will allow a variable number of arguments in its invocation. This class cannot be inherited.
PlatformNotSupportedException	The exception that is thrown when a feature does not run on a particular platform.
Progress<T>	Provides an IProgress<T> that invokes callbacks for each reported progress value.
Random	Represents a pseudo-random number generator, which is an algorithm that produces a sequence of numbers that meet certain statistical requirements for randomness.
RankException	The exception that is thrown when an array with the wrong number of dimensions is passed to a method.
ResolveEventArgs	Provides data for loader resolution events, such as the TypeResolve, ResourceResolve, ReflectionOnlyAssemblyResolve, and AssemblyResolve events.
SerializableAttribute	Indicates that a class can be serialized using binary or XML serialization. This class cannot be inherited.
StackOverflowException	The exception that is thrown when the execution stack exceeds the stack size. This class cannot be inherited.
STAThreadAttribute	Indicates that the COM threading model for an application is single-threaded apartment (STA).
String	Represents text as a sequence of UTF-16 code units.
StringComparer	Represents a string comparison operation that uses specific case and culture-based or ordinal comparison rules.
StringNormalizationExtensions	Provides extension methods to work with string normalization.
SystemException	Serves as the base class for system exceptions namespace.
ThreadStaticAttribute	Indicates that the value of a static field is unique for each thread.
TimeoutException	The exception that is thrown when the time allotted for a process or operation has expired.
TimeProvider	Provides an abstraction for time.

TimeZone	Represents a time zone.
TimeZoneInfo	Represents any time zone in the world.
TimeZoneInfo.AdjustmentRule	Provides information about a time zone adjustment, such as the transition to and from daylight saving time.
TimeZoneNotFoundException	The exception that is thrown when a time zone cannot be found.
Tuple	Provides static methods for creating tuple objects.
Tuple<T1>	Represents a 1-tuple, or singleton.
Tuple<T1,T2>	Represents a 2-tuple, or pair.
Tuple<T1,T2,T3>	Represents a 3-tuple, or triple.
Tuple<T1,T2,T3,T4>	Represents a 4-tuple, or quadruple.
Tuple<T1,T2,T3,T4,T5>	Represents a 5-tuple, or quintuple.
Tuple<T1,T2,T3,T4,T5,T6>	Represents a 6-tuple, or sextuple.
Tuple<T1,T2,T3,T4,T5,T6,T7>	Represents a 7-tuple, or septuple.
Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>	Represents an *n*-tuple, where *n* is 8 or greater.
TupleExtensions	Provides extension methods for tuples to interoperate with language support for tuples in C#.
Type	Represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types.
TypeAccessException	The exception that is thrown when a method attempts to use a type that it does not have access to.
TypeInitializationException	The exception that is thrown as a wrapper around the exception thrown by the class initializer. This class cannot be inherited.
TypeLoadException	The exception that is thrown when type-loading failures occur.
TypeUnloadedException	The exception that is thrown when there is an attempt to access an unloaded class.
UnauthorizedAccessException	The exception that is thrown when the operating system denies access because of an I/O error or a specific type of security error.

UnhandledExceptionEventArgs	Provides data for the event that is raised when there is an exception that is not handled in any application domain.
Uri	Provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI.
UriBuilder	Provides a custom constructor for uniform resource identifiers (URIs) and modifies URIs for the Uri class.
UriFormatException	The exception that is thrown when an invalid Uniform Resource Identifier (URI) is detected.
UriParser	Parses a new URI scheme. This is an abstract class.
UriTypeConverter	Converts a String type to a Uri type, and vice versa.
ValueType	Provides the base class for value types.
Version	Represents the version number of an assembly, operating system, or the common language runtime. This class cannot be inherited.
WeakReference	Represents a weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.
WeakReference<T>	Represents a typed weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.

Structs

Expand table

ArgIterator	Represents a variable-length argument list; that is, the parameters of a function that takes a variable number of arguments.
ArraySegment<T>.Enumerator	Provides an enumerator for the elements of an ArraySegment<T>.
ArraySegment<T>	Delimits a section of a one-dimensional array.
Boolean	Represents a Boolean (`true` or `false`) value.
Byte	Represents an 8-bit unsigned integer.
Char	Represents a character as a UTF-16 code unit.

ConsoleKeyInfo	Describes the console key that was pressed, including the character represented by the console key and the state of the SHIFT, ALT, and CTRL modifier keys.
DateOnly	Represents dates with values ranging from January 1, 0001 Anno Domini (Common Era) through December 31, 9999 A.D. (C.E.) in the Gregorian calendar.
DateTime	Represents an instant in time, typically expressed as a date and time of day.
DateTimeOffset	Represents a point in time, typically expressed as a date and time of day, relative to Coordinated Universal Time (UTC).
Decimal	Represents a decimal floating-point number.
Double	Represents a double-precision floating-point number.
GCGenerationInfo	Represents the size and the fragmenation of a generation on entry and on exit of the GC reported in GCMemoryInfo.
GCMemoryInfo	Provides a set of APIs that can be used to retrieve garbage collection information.
Guid	Represents a globally unique identifier (GUID).
Half	Represents a half-precision floating-point number.
HashCode	Combines the hash code for multiple values into a single hash code.
Index	Represents a type that can be used to index a collection either from the beginning or the end.
Int128	Represents a 128-bit signed integer.
Int16	Represents a 16-bit signed integer.
Int32	Represents a 32-bit signed integer.
Int64	Represents a 64-bit signed integer.
IntPtr	Represents a signed integer where the bit-width is the same as a pointer.
Memory<T>	Represents a contiguous region of memory.
MemoryExtensions.TryWriteInterpolatedStringHandler	Provides a handler used by the language compiler to format interpolated strings into character spans.

ModuleHandle	Represents a runtime handle for a module.
Nullable<T>	Represents a value type that can be assigned `null`.
Range	Represents a range that has start and end indexes.
ReadOnlyMemory<T>	Represents a contiguous region of memory, similar to ReadOnlySpan<T>. Unlike ReadOnlySpan<T>, it is not a byref-like type.
ReadOnlySpan<T>.Enumerator	Provides an enumerator for the elements of a ReadOnlySpan<T>.
ReadOnlySpan<T>	Provides a type-safe and memory-safe read-only representation of a contiguous region of arbitrary memory.
RuntimeArgumentHandle	References a variable-length argument list.
RuntimeFieldHandle	Represents a field using an internal metadata token.
RuntimeMethodHandle	RuntimeMethodHandle is a handle to the internal metadata representation of a method.
RuntimeTypeHandle	Represents a type using an internal metadata token.
SByte	Represents an 8-bit signed integer.
SequencePosition	Represents a position in a non-contiguous set of memory. Properties of this type should not be interpreted by anything but the type that created it.
Single	Represents a single-precision floating-point number.
Span<T>.Enumerator	Provides an enumerator for the elements of a Span<T>.
Span<T>	Provides a type-safe and memory-safe representation of a contiguous region of arbitrary memory.
TimeOnly	Represents a time of day, as would be read from a clock, within the range 00:00:00 to 23:59:59.9999999.
TimeSpan	Represents a time interval.
TimeZoneInfo.TransitionTime	Provides information about a specific time change, such as the change from daylight saving time to standard time or vice versa, in a particular time zone.
TypedReference	Describes objects that contain both a managed pointer to a location and a runtime representation of the type that may be stored at that location.

UInt128	Represents a 128-bit unsigned integer.
UInt16	Represents a 16-bit unsigned integer.
UInt32	Represents a 32-bit unsigned integer.
UInt64	Represents a 64-bit unsigned integer.
UIntPtr	Represents an unsigned integer where the bit-width is the same as a pointer.
UriCreationOptions	Provides options that control how a Uri is created and behaves.
ValueTuple	Provides static methods for creating value tuples.
ValueTuple<T1>	Represents a value tuple with a single component.
ValueTuple<T1,T2>	Represents a value tuple with 2 components.
ValueTuple<T1,T2,T3>	Represents a value tuple with 3 components.
ValueTuple<T1,T2,T3,T4>	Represents a value tuple with 4 components.
ValueTuple<T1,T2,T3,T4,T5>	Represents a value tuple with 5 components.
ValueTuple<T1,T2,T3,T4,T5,T6>	Represents a value tuple with 6 components.
ValueTuple<T1,T2,T3,T4,T5,T6,T7>	Represents a value tuple with 7 components.
ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>	Represents an *n*-value tuple, where *n* is 8 or greater.
Void	Specifies a return value type for a method that does not return a value.

Interfaces

Expand table

IAsyncDisposable	Provides a mechanism for releasing unmanaged resources asynchronously.
IAsyncResult	Represents the status of an asynchronous operation.
ICloneable	Supports cloning, which creates a new instance of a class with the same value as an existing instance.
IComparable	Defines a generalized type-specific comparison method that a value type or class implements to order or sort its instances.

IComparable<T>	Defines a generalized comparison method that a value type or class implements to create a type-specific comparison method for ordering or sorting its instances.
IConvertible	Defines methods that convert the value of the implementing reference or value type to a common language runtime type that has an equivalent value.
ICustomFormatter	Defines a method that supports custom formatting of the value of an object.
IDisposable	Provides a mechanism for releasing unmanaged resources.
IEquatable<T>	Defines a generalized method that a value type or class implements to create a type-specific method for determining equality of instances.
IFormatProvider	Provides a mechanism for retrieving an object to control formatting.
IFormattable	Provides functionality to format the value of an object into a string representation.
IObservable<T>	Defines a provider for push-based notification.
IObserver<T>	Provides a mechanism for receiving push-based notifications.
IParsable<TSelf>	Defines a mechanism for parsing a string to a value.
IProgress<T>	Defines a provider for progress updates.
IServiceProvider	Defines a mechanism for retrieving a service object; that is, an object that provides custom support to other objects.
ISpanFormattable	Provides functionality to format the string representation of an object into a span.
ISpanParsable<TSelf>	Defines a mechanism for parsing a span of characters to a value.
IUtf8SpanFormattable	Provides functionality to format the string representation of an object into a span as UTF-8.
IUtf8SpanParsable<TSelf>	Defines a mechanism for parsing a span of UTF-8 characters to a value.

Enums

Expand table

AttributeTargets	Specifies the application elements on which it is valid to apply an

	attribute.
Base64FormattingOptions	Specifies whether relevant ToBase64CharArray and ToBase64String methods insert line breaks in their output.
ConsoleColor	Specifies constants that define foreground and background colors for the console.
ConsoleKey	Specifies the standard keys on a console.
ConsoleModifiers	Represents the SHIFT, ALT, and CTRL modifier keys on a keyboard.
ConsoleSpecialKey	Specifies combinations of modifier and console keys that can interrupt the current process.
DateTimeKind	Specifies whether a DateTime object represents a local time, a Coordinated Universal Time (UTC), or is not specified as either local time or UTC.
DayOfWeek	Specifies the day of the week.
Environment.SpecialFolder	Specifies enumerated constants used to retrieve directory paths to system special folders.
Environment.SpecialFolderOption	Specifies options to use for getting the path to a special folder.
EnvironmentVariableTarget	Specifies the location where an environment variable is stored or retrieved in a set or get operation.
GCCollectionMode	Specifies the behavior for a forced garbage collection.
GCKind	Specifies the kind of a garbage collection.
GCNotificationStatus	Provides information about the current registration for notification of the next full garbage collection.
GenericUriParserOptions	Specifies options for a UriParser.
LoaderOptimization	An enumeration used with the LoaderOptimizationAttribute class to specify loader optimizations for an executable.
MidpointRounding	Specifies the strategy that mathematical rounding methods should use to round a number.
PlatformID	Identifies the operating system, or platform, supported by an assembly.
StringComparison	Specifies the culture, case, and sort rules to be used by certain overloads of the Compare(String, String) and Equals(Object) methods.
StringSplitOptions	Specifies options for applicable Split method overloads, such as whether to omit empty substrings from the returned array or trim

	whitespace from substrings.
TypeCode	Specifies the type of an object.
UriComponents	Specifies the parts of a Uri.
UriFormat	Controls how URI information is escaped.
UriHostNameType	Defines host name types for the CheckHostName(String) method.
UriKind	Defines the different kinds of URIs.
UriPartial	Defines the parts of a URI for the GetLeftPart(UriPartial) method.

Delegates

Expand table

Action	Encapsulates a method that has no parameters and does not return a value.
Action<T>	Encapsulates a method that has a single parameter and does not return a value.
Action<T1,T2>	Encapsulates a method that has two parameters and does not return a value.
Action<T1,T2,T3>	Encapsulates a method that has three parameters and does not return a value.
Action<T1,T2,T3,T4>	Encapsulates a method that has four parameters and does not return a value.
Action<T1,T2,T3,T4,T5>	Encapsulates a method that has five parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6>	Encapsulates a method that has six parameters

	and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7>	Encapsulates a method that has seven parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8>	Encapsulates a method that has eight parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9>	Encapsulates a method that has nine parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>	Encapsulates a method that has 10 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>	Encapsulates a method that has 11 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>	Encapsulates a method that has 12 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>	Encapsulates a method that has 13 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>	Encapsulates a method that has 14 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>	Encapsulates a method that has 15 parameters and does not return a value.
Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>	Encapsulates a method that has 16 parameters

	and does not return a value.
AssemblyLoadEventHandler	Represents the method that handles the AssemblyLoad event of an AppDomain.
AsyncCallback	References a method to be called when a corresponding asynchronous operation completes.
Comparison<T>	Represents the method that compares two objects of the same type.
ConsoleCancelEventHandler	Represents the method that will handle the CancelKeyPress event of a Console.
Converter<TInput,TOutput>	Represents a method that converts an object from one type to another type.
EventHandler	Represents the method that will handle an event that has no event data.
EventHandler<TEventArgs>	Represents the method that will handle an event when the event provides data.
Func<TResult>	Encapsulates a method that has no parameters and returns a value of the type specified by the `TResult` parameter.
Func<T,TResult>	Encapsulates a method that has one parameter and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,TResult>	Encapsulates a method that has two parameters

	and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,TResult>	Encapsulates a method that has three parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,TResult>	Encapsulates a method that has four parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,TResult>	Encapsulates a method that has five parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,TResult>	Encapsulates a method that has six parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,TResult>	Encapsulates a method that has seven parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult>	Encapsulates a method that has eight parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult>	Encapsulates a method that has nine parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult>	Encapsulates a method that has 10 parameters

	and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult>	Encapsulates a method that has 11 parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult>	Encapsulates a method that has 12 parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult>	Encapsulates a method that has 13 parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult>	Encapsulates a method that has 14 parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult>	Encapsulates a method that has 15 parameters and returns a value of the type specified by the `TResult` parameter.
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult>	Encapsulates a method that has 16 parameters and returns a value of the type specified by the `TResult` parameter.
Predicate<T>	Represents the method that defines a set of criteria and determines whether the specified object meets those criteria.
ResolveEventHandler	Represents a method that handles the TypeResolve, ResourceResolve, or

	AssemblyResolve event of an AppDomain.
UnhandledExceptionEventHandler	Represents the method that will handle the event raised by an exception that is not handled by the application domain.

Remarks

Other classes provide services supporting data type conversion, method parameter manipulation, mathematics, remote and local program invocation, application environment management, and supervision of managed and unmanaged applications.

To access types in the System namespace from C# code without having to provide fully qualified type names, include the `using System;` statement at the beginning of your source code file.

To access types in the System namespace from C++/CLI code without having to provide fully qualified type names, include the `using namespace System;` statement at the beginning of your source code file. In addition, you must compile with the /clr switch.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AccessViolationException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to read or write protected memory.

C#

```
public class AccessViolationException : SystemException
```

Inheritance Object → Exception → SystemException → AccessViolationException

Remarks

For more information about this API, see Supplemental API remarks for AccessViolationException.

Constructors

Expand table

AccessViolationException()	Initializes a new instance of the AccessViolationException class with a system-supplied message that describes the error.
AccessViolationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AccessViolationException class with serialized data.
AccessViolationException(String)	Initializes a new instance of the AccessViolationException class with a specified message that describes the error.
AccessViolationException(String, Exception)	Initializes a new instance of the AccessViolationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

AccessViolationException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the AccessViolationException class.

Overloads

Expand table

AccessViolationException()	Initializes a new instance of the AccessViolationException class with a system-supplied message that describes the error.
AccessViolationException(String)	Initializes a new instance of the AccessViolationException class with a specified message that describes the error.
AccessViolationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AccessViolationException class with serialized data.
AccessViolationException(String, Exception)	Initializes a new instance of the AccessViolationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

AccessViolationException()

Initializes a new instance of the AccessViolationException class with a system-supplied message that describes the error.

C#

```
public AccessViolationException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to read or write protected memory. This is often an indication that other memory has been corrupted." This message takes into account the current system culture.

The following table shows the initial property values for an instance of AccessViolationException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AccessViolationException(String)

Initializes a new instance of the AccessViolationException class with a specified message that describes the error.

C#

```
public AccessViolationException (string? message);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of AccessViolationException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AccessViolationException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the AccessViolationException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected AccessViolationException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

.NET Framework	2.0, 3.0, 3.5, 3.5 Client Profile, 4, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AccessViolationException(String, Exception)

Initializes a new instance of the AccessViolationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public AccessViolationException (string? message, Exception?
innerException);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of AccessViolationException.

Expand table

Property	Value
InnerException	`null`.

Property	Value
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has no parameters and does not return a value.

C#

```
public delegate void Action();
```

Remarks

You can use this delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have no parameters and no return value. (In C#, the method must return `void`. In F# the function or method must return `unit`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has no parameters and returns a value, use the generic **Func<TResult>** delegate instead.

When you use the Action delegate, you do not have to explicitly define a delegate that encapsulates a parameterless procedure. For example, the following code explicitly declares a delegate named `ShowValue` and assigns a reference to the `Name.DisplayToWindow` instance method to its delegate instance.

C#

```
using System;
using System.Windows.Forms;

public delegate void ShowValue();
```

```
public class Name
{
   private string instanceName;

   public Name(string name)
   {
      this.instanceName = name;
   }

   public void DisplayToConsole()
   {
      Console.WriteLine(this.instanceName);
   }

   public void DisplayToWindow()
   {
      MessageBox.Show(this.instanceName);
   }
}

public class testTestDelegate
{
   public static void Main()
   {
      Name testName = new Name("Koani");
      ShowValue showMethod = testName.DisplayToWindow;
      showMethod();
   }
}
```

The following example simplifies this code by instantiating the Action delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;
using System.Windows.Forms;

public class Name
{
   private string instanceName;

   public Name(string name)
   {
      this.instanceName = name;
   }

   public void DisplayToConsole()
   {
      Console.WriteLine(this.instanceName);
   }
```

```
    public void DisplayToWindow()
    {
        MessageBox.Show(this.instanceName);
    }
}

public class testTestDelegate
{
    public static void Main()
    {
        Name testName = new Name("Koani");
        Action showMethod = testName.DisplayToWindow;
        showMethod();
    }
}
```

You can also use the Action delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;
using System.Windows.Forms;

public class Name
{
    private string instanceName;

    public Name(string name)
    {
        this.instanceName = name;
    }

    public void DisplayToConsole()
    {
        Console.WriteLine(this.instanceName);
    }

    public void DisplayToWindow()
    {
        MessageBox.Show(this.instanceName);
    }
}

public class Anonymous
{
    public static void Main()
    {
        Name testName = new Name("Koani");
        Action showMethod = delegate() { testName.DisplayToWindow();} ;
        showMethod();
```

```
    }
}
```

You can also assign a lambda expression to an Action delegate instance, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (C#) or Lambda Expressions (F#).)

C#

```
using System;
using System.Windows.Forms;

public class Name
{
    private string instanceName;

    public Name(string name)
    {
        this.instanceName = name;
    }

    public void DisplayToConsole()
    {
        Console.WriteLine(this.instanceName);
    }

    public void DisplayToWindow()
    {
        MessageBox.Show(this.instanceName);
    }
}

public class LambdaExpression
{
    public static void Main()
    {
        Name testName = new Name("Koani");
        Action showMethod = () => testName.DisplayToWindow();
        showMethod();
    }
}
```

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has a single parameter and does not return a value.

```
C#
public delegate void Action<in T>(T obj);
```

Type Parameters

T

The type of the parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

obj T

The parameter of the method that this delegate encapsulates.

Examples

The following example demonstrates the use of the Action<T> delegate to print the contents of a List<T> object. In this example, the `Print` method is used to display the contents of the list to the console. In addition, the C# example also demonstrates the use of anonymous methods to display the contents to the console. Note that the example does not explicitly declare an Action<T> variable. Instead, it passes a reference to a method that takes a single parameter and that does not return a value to the List<T>.ForEach method, whose single parameter is an Action<T> delegate. Similarly, in the C# example, an Action<T> delegate is not explicitly instantiated because the signature of the anonymous method matches the signature of the Action<T> delegate that is expected by the List<T>.ForEach method.

C#

```
List<string> names = new List<string>();
names.Add("Bruce");
names.Add("Alfred");
names.Add("Tim");
names.Add("Richard");

// Display the contents of the list using the Print method.
names.ForEach(Print);

// The following demonstrates the anonymous method feature of C#
// to display the contents of the list to the console.
names.ForEach(delegate(string name)
{
    Console.WriteLine(name);
});

void Print(string s)
{
    Console.WriteLine(s);
}

/* This code will produce output similar to the following:
* Bruce
* Alfred
* Tim
* Richard
* Bruce
* Alfred
* Tim
* Richard
*/
```

Remarks

You can use the Action<T> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have one parameter that is passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

ⓘ **Note**

To reference a method that has one parameter and returns a value, use the generic **Func<T,TResult>** delegate instead.

When you use the Action<T> delegate, you do not have to explicitly define a delegate that encapsulates a method with a single parameter. For example, the following code explicitly declares a delegate named `DisplayMessage` and assigns a reference to either the WriteLine method or the `ShowWindowsMessage` method to its delegate instance.

C#

```
using System;
using System.Windows.Forms;

delegate void DisplayMessage(string message);

public class TestCustomDelegate
{
   public static void Main()
   {
      DisplayMessage messageTarget;

      if (Environment.GetCommandLineArgs().Length > 1)
         messageTarget = ShowWindowsMessage;
      else
         messageTarget = Console.WriteLine;

      messageTarget("Hello, World!");
   }

   private static void ShowWindowsMessage(string message)
   {
      MessageBox.Show(message);
   }
}
```

The following example simplifies this code by instantiating the Action<T> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;
using System.Windows.Forms;

public class TestAction1
{
   public static void Main()
   {
      Action<string> messageTarget;
```

```
        if (Environment.GetCommandLineArgs().Length > 1)
            messageTarget = ShowWindowsMessage;
        else
            messageTarget = Console.WriteLine;

        messageTarget("Hello, World!");
    }

    private static void ShowWindowsMessage(string message)
    {
        MessageBox.Show(message);
    }
}
```

You can also use the Action<T> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;
using System.Windows.Forms;

public class TestAnonMethod
{
    public static void Main()
    {
        Action<string> messageTarget;

        if (Environment.GetCommandLineArgs().Length > 1)
            messageTarget = delegate(string s) { ShowWindowsMessage(s); };
        else
            messageTarget = delegate(string s) { Console.WriteLine(s); };

        messageTarget("Hello, World!");
    }

    private static void ShowWindowsMessage(string message)
    {
        MessageBox.Show(message);
    }
}
```

You can also assign a lambda expression to an Action<T> delegate instance, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions.)

C#

```
using System;
using System.Windows.Forms;

public class TestLambdaExpression
{
   public static void Main()
   {
      Action<string> messageTarget;

      if (Environment.GetCommandLineArgs().Length > 1)
         messageTarget = s => ShowWindowsMessage(s);
      else
         messageTarget = s => Console.WriteLine(s);

      messageTarget("Hello, World!");
   }

   private static void ShowWindowsMessage(string message)
   {
      MessageBox.Show(message);
   }
}
```

The ForEach and ForEach methods each take an Action<T> delegate as a parameter. The method encapsulated by the delegate allows you to perform an action on each element in the array or list. The example uses the ForEach method to provide an illustration.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has two parameters and does not return a value.

```
C#
public delegate void Action<in T1,in T2>(T1 arg1, T2 arg2);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have two parameters that are both passed to it by value, and it must not return a value. (In C#, the method must return `void`. In F#, the method or function must return unit. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has two parameters and returns a value, use the generic **Func<T1,T2,TResult>** delegate instead.

When you use the Action<T1,T2> delegate, you do not have to explicitly define a delegate that encapsulates a method with two parameters. For example, the following code explicitly declares a delegate named `ConcatStrings`. It then assigns a reference to either of two methods to its delegate instance. One method writes two strings to the console; the second writes two strings to a file.

C#

```
using System;
using System.IO;

delegate void ConcatStrings(string string1, string string2);

public class TestDelegate
{
   public static void Main()
   {
      string message1 = "The first line of a message.";
      string message2 = "The second line of a message.";
      ConcatStrings concat;

      if (Environment.GetCommandLineArgs().Length > 1)
         concat = WriteToFile;
      else
         concat = WriteToConsole;

      concat(message1, message2);
   }

   private static void WriteToConsole(string string1, string string2)
   {
      Console.WriteLine("{0}\n{1}", string1, string2);
   }
```

```
   private static void WriteToFile(string string1, string string2)
   {
      StreamWriter writer = null;
      try
      {
         writer = new StreamWriter(Environment.GetCommandLineArgs()[1],
false);
         writer.WriteLine("{0}\n{1}", string1, string2);
      }
      catch
      {
         Console.WriteLine("File write operation failed...");
      }
      finally
      {
         if (writer != null) writer.Close();
      }
   }
}
```

The following example simplifies this code by instantiating the Action<T1,T2> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;
using System.IO;

public class TestAction2
{
   public static void Main()
   {
      string message1 = "The first line of a message.";
      string message2 = "The second line of a message.";
      Action<string, string> concat;

      if (Environment.GetCommandLineArgs().Length > 1)
         concat = WriteToFile;
      else
         concat = WriteToConsole;

      concat(message1, message2);
   }

   private static void WriteToConsole(string string1, string string2)
   {
      Console.WriteLine("{0}\n{1}", string1, string2);
   }

   private static void WriteToFile(string string1, string string2)
   {
      StreamWriter writer = null;
```

```
      try
      {
         writer = new StreamWriter(Environment.GetCommandLineArgs()[1],
false);
         writer.WriteLine("{0}\n{1}", string1, string2);
      }
      catch
      {
         Console.WriteLine("File write operation failed...");
      }
      finally
      {
         if (writer != null) writer.Close();
      }
   }
}
```

You can also use the Action<T1,T2> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;
using System.IO;

public class TestAnonymousMethod
{
   public static void Main()
   {
      string message1 = "The first line of a message.";
      string message2 = "The second line of a message.";
      Action<string, string> concat;

      if (Environment.GetCommandLineArgs().Length > 1)
         concat = delegate(string s1, string s2) { WriteToFile(s1, s2); };
      else
         concat = delegate(string s1, string s2) { WriteToConsole(s1, s2);}
;

      concat(message1, message2);
   }

   private static void WriteToConsole(string string1, string string2)
   {
      Console.WriteLine("{0}\n{1}", string1, string2);
   }

   private static void WriteToFile(string string1, string string2)
   {
      StreamWriter writer = null;
      try
```

```
      {
         writer = new StreamWriter(Environment.GetCommandLineArgs()[1],
false);
         writer.WriteLine("{0}\n{1}", string1, string2);
      }
      catch
      {
         Console.WriteLine("File write operation failed...");
      }
      finally
      {
         if (writer != null) writer.Close();
      }
   }
}
```

You can also assign a lambda expression to an Action<T1,T2> delegate instance, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (C#), or Lambda Expressions (F#).)

C#

```
using System;
using System.IO;

public class TestLambdaExpression
{
   public static void Main()
   {
      string message1 = "The first line of a message.";
      string message2 = "The second line of a message.";
      Action<string, string> concat;

      if (Environment.GetCommandLineArgs().Length > 1)
         concat = (s1, s2) => WriteToFile(s1, s2);
      else
         concat = (s1, s2) => WriteToConsole(s1, s2);

      concat(message1, message2);
   }

   private static void WriteToConsole(string string1, string string2)
   {
      Console.WriteLine("{0}\n{1}", string1, string2);
   }

   private static void WriteToFile(string string1, string string2)
   {
      StreamWriter writer = null;
      try
      {
         writer = new StreamWriter(Environment.GetCommandLineArgs()[1],
```

```
false);
            writer.WriteLine("{0}\n{1}", string1, string2);
        }
        catch
        {
            Console.WriteLine("File write operation failed...");
        }
        finally
        {
            if (writer != null) writer.Close();
        }
    }
}
```

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Action<T1,T2,T3> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has three parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3>(T1 arg1, T2 arg2, T3 arg3);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

arg3 T3

The third parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have three parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In F#, the method or function must return unit. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> ⓘ **Note**
>
> To reference a method that has three parameters and returns a value, use the generic **Func<T1,T2,T3,TResult>** delegate instead.

When you use the Action<T1,T2,T3> delegate, you do not have to explicitly define a delegate that encapsulates a method with three parameters. For example, the following code explicitly declares a delegate named `StringCopy` and assigns a reference to the `CopyStrings` method to its delegate instance.

C#

```
using System;

delegate void StringCopy(string[] stringArray1,
                         string[] stringArray2,
                         int indexToStart);

public class TestDelegate
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      StringCopy copyOperation = CopyStrings;
      copyOperation(ordinals, copiedOrdinals, 3);
      foreach (string ordinal in copiedOrdinals)
```

```
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target, int
startPos)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= source.Length - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

The following example simplifies this code by instantiating the Action<T1,T2,T3> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;

public class TestAction3
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int> copyOperation = CopyStrings;
      copyOperation(ordinals, copiedOrdinals, 3);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target, int
startPos)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= source.Length - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

You can also use the Action<T1,T2,T3> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see

Anonymous Methods.)

C#

```
using System;

public class TestAnon
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int> copyOperation = delegate(string[] s1,
                                                               string[] s2,
                                                               int pos)
                                     { CopyStrings(s1, s2, pos); };
      copyOperation(ordinals, copiedOrdinals, 3);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target, int
startPos)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= source.Length - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

You can also assign a lambda expression to an Action<T1,T2,T3> delegate instance, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (C#) or Lambda Expressions (F#).)

C#

```
using System;

public class TestLambda
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int> copyOperation = (s1, s2, pos) =>
                                     CopyStrings(s1, s2, pos);
      copyOperation(ordinals, copiedOrdinals, 3);
      foreach (string ordinal in copiedOrdinals)
```

```
            Console.WriteLine(ordinal == string.Empty ? "<None>" : ordinal);
    }

    private static void CopyStrings(string[] source, string[] target, int
startPos)
    {
        if (source.Length != target.Length)
            throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

        for (int ctr = startPos; ctr <= source.Length - 1; ctr++)
            target[ctr] = string.Copy(source[ctr]);
    }
}
```

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has four parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4>(T1 arg1, T2 arg2, T3
arg3, T4 arg4);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have four parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In F#, the method or function must return unit. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has four parameters and returns a value, use the generic **Func<T1,T2,T3,T4,TResult>** delegate instead.

When you use the Action<T1,T2,T3,T4> delegate, you do not have to explicitly define a delegate that encapsulates a method with four parameters. For example, the following code explicitly declares a delegate named `StringCopy` and assigns a reference to the `CopyStrings` method to its delegate instance.

```
C#
```

```
using System;

delegate void StringCopy(string[] stringArray1,
                         string[] stringArray2,
                         int indexToStart,
                         int numberToCopy);

public class TestDelegate
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth",
                           "Sixth", "Seventh", "Eighth", "Ninth", "Tenth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      StringCopy copyOperation = CopyStrings;
      copyOperation(ordinals, copiedOrdinals, 3, 5);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target,
                                   int startPos, int number)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= startPos + number - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

The following example simplifies this code by instantiating the Action<T1,T2,T3,T4> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;

public class TestAction4
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth",
                           "Sixth", "Seventh", "Eighth", "Ninth", "Tenth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int, int> copyOperation = CopyStrings;
      copyOperation(ordinals, copiedOrdinals, 3, 5);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
```

```
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target,
                                   int startPos, int number)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= startPos + number - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

You can also use the Action<T1,T2,T3,T4> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;

public class TestAnonymousMethod
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth",
                           "Sixth", "Seventh", "Eighth", "Ninth", "Tenth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int, int> copyOperation =
                                           delegate(string[] s1, string[]
s2,
                                           int pos, int num)
                                 { CopyStrings(s1, s2, pos, num); };
      copyOperation(ordinals, copiedOrdinals, 3, 5);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target,
                                   int startPos, int number)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= startPos + number - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

You can also assign a lambda expression to an Action<T1,T2,T3,T4> delegate instance, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (C#) or Lambda Expressions (F#).)

C#

```
using System;

public class TestLambdaExpression
{
   public static void Main()
   {
      string[] ordinals = {"First", "Second", "Third", "Fourth", "Fifth",
                           "Sixth", "Seventh", "Eighth", "Ninth", "Tenth"};
      string[] copiedOrdinals = new string[ordinals.Length];
      Action<string[], string[], int, int> copyOperation = (s1, s2, pos,
num)
                                              => CopyStrings(s1, s2, pos, num);
      copyOperation(ordinals, copiedOrdinals, 3, 5);
      foreach (string ordinal in copiedOrdinals)
         Console.WriteLine(string.IsNullOrEmpty(ordinal) ? "<None>" :
ordinal);
   }

   private static void CopyStrings(string[] source, string[] target,
                                   int startPos, int number)
   {
      if (source.Length != target.Length)
         throw new IndexOutOfRangeException("The source and target arrays
must have the same number of elements.");

      for (int ctr = startPos; ctr <= startPos + number - 1; ctr++)
         target[ctr] = string.Copy(source[ctr]);
   }
}
```

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has five parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5>(T1 arg1, T2 arg2,
T3 arg3, T4 arg4, T5 arg5);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

T5

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

arg3 T3

The third parameter of the method that this delegate encapsulates.

arg4 T4

The fourth parameter of the method that this delegate encapsulates.

arg5 T5

The fifth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have five parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has five parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has six parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have six parameters that are all passed to it by value,

and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has six parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has seven parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7>(T1
arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

T5

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T6

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T7

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

arg3 T3

The third parameter of the method that this delegate encapsulates.

arg4 T4

The fourth parameter of the method that this delegate encapsulates.

arg5 T5

The fifth parameter of the method that this delegate encapsulates.

arg6 T6

The sixth parameter of the method that this delegate encapsulates.

arg7 T7

The seventh parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have seven parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has seven parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has eight parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8>
(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T5

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T6

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T7

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T8

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

arg3 T3

The third parameter of the method that this delegate encapsulates.

arg4 T4

The fourth parameter of the method that this delegate encapsulates.

arg5 T5

The fifth parameter of the method that this delegate encapsulates.

arg6 T6

The sixth parameter of the method that this delegate encapsulates.

arg7 T7

The seventh parameter of the method that this delegate encapsulates.

arg8 T8

The eighth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have eight parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has eight parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has nine parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,in T9>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8
arg8, T9 arg9);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T5

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T6

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T7

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T8

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T9

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

arg3 T3

The third parameter of the method that this delegate encapsulates.

arg4 T4

The fourth parameter of the method that this delegate encapsulates.

arg5 T5

The fifth parameter of the method that this delegate encapsulates.

arg6 T6

The sixth parameter of the method that this delegate encapsulates.

arg7 T7

The seventh parameter of the method that this delegate encapsulates.

arg8 T8

The eighth parameter of the method that this delegate encapsulates.

arg9 T9

The ninth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have nine parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has nine parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 10 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,in T9,in T10>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7
arg7, T8 arg8, T9 arg9, T10 arg10);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 10 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that

returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has 10 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 11 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 11 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has 11 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 12 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 12 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has 12 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 13 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 13 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> ⓘ **Note**
>
> To reference a method that has 13 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 14 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,in T14>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13, T14 arg14);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T14`

The type of the fourteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

arg6 T6

The sixth parameter of the method that this delegate encapsulates.

arg7 T7

The seventh parameter of the method that this delegate encapsulates.

arg8 T8

The eighth parameter of the method that this delegate encapsulates.

arg9 T9

The ninth parameter of the method that this delegate encapsulates.

arg10 T10

The tenth parameter of the method that this delegate encapsulates.

arg11 T11

The eleventh parameter of the method that this delegate encapsulates.

arg12 T12

The twelfth parameter of the method that this delegate encapsulates.

arg13 T13

The thirteenth parameter of the method that this delegate encapsulates.

arg14 T14

The fourteenth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 14 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

Note

To reference a method that has 14 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 15 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,in T14,in T15>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13, T14 arg14, T15 arg15);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T14`

The type of the fourteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T15`

The type of the fifteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

`arg14` T14

The fourteenth parameter of the method that this delegate encapsulates.

`arg15` T15

The fifteenth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 15 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has 15 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 16 parameters and does not return a value.

C#

```
public delegate void Action<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,in T9,in T10,in T11,in T12,in T13,in T14,in T15,in T16>(T1 arg1, T2 arg2,
T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10,
T11 arg11, T12 arg12, T13 arg13, T14 arg14, T15 arg15, T16 arg16);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T11

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T12

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T13

The type of the thirteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T14

The type of the fourteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T15

The type of the fifteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T16

The type of the sixteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

arg1 T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

`arg14` T14

The fourteenth parameter of the method that this delegate encapsulates.

`arg15` T15

The fifteenth parameter of the method that this delegate encapsulates.

`arg16` T16

The sixteenth parameter of the method that this delegate encapsulates.

Remarks

You can use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> delegate to pass a method as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 16 parameters that are all passed to it by value, and it must not return a value. (In C#, the method must return `void`. In Visual Basic, it must be defined by the `Sub`...`End Sub` construct. It can also be a method that returns a value that is ignored.) Typically, such a method is used to perform an operation.

> **Note**
>
> To reference a method that has 16 parameters and returns a value, use the generic **Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult>** delegate instead.

You can also use the Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> delegate with anonymous methods and lambda expressions.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Activator Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Contains methods to create types of objects locally or remotely, or obtain references to existing remote objects. This class cannot be inherited.

C#

```
public static class Activator
```

Inheritance Object → Activator

Examples

The following example shows how to use the Activator class to dynamically construct objects at run time.

C#

```
using System;
using System.Reflection;
using System.Text;

public class SomeType
{
    public void DoSomething(int x)
    {
        Console.WriteLine("100 / {0} = {1}", x, 100 / x);
    }
}

public class Example
{
    static void Main()
    {
        // Create an instance of the StringBuilder type using
        // Activator.CreateInstance.
        Object o = Activator.CreateInstance(typeof(StringBuilder));

        // Append a string into the StringBuilder object and display the
```

```
        // StringBuilder.
        StringBuilder sb = (StringBuilder) o;
        sb.Append("Hello, there.");
        Console.WriteLine(sb);

        // Create an instance of the SomeType class that is defined in this
        // assembly.
        System.Runtime.Remoting.ObjectHandle oh =

Activator.CreateInstanceFrom(Assembly.GetEntryAssembly().CodeBase,
                                        typeof(SomeType).FullName);

        // Call an instance method defined by the SomeType type using this
object.
        SomeType st = (SomeType) oh.Unwrap();

        st.DoSomething(5);
    }
}

/* This code produces the following output:

Hello, there.
100 / 5 = 20
 */
```

Remarks

The CreateInstance method creates an instance of a type defined in an assembly by invoking the constructor that best matches the specified arguments. If no arguments are specified, the constructor that takes no parameters, that is, the parameterless constructor, is invoked.

You must have sufficient permission to search for and call a constructor; otherwise, an exception is thrown. By default, only public constructors are considered during the search for a constructor. If no constructor or parameterless constructor can be found, an exception is thrown.

A binder parameter specifies an object that searches an assembly for a suitable constructor. You can specify your own binder and search criteria. If no binder is specified, a default binder is used. For more information, see the System.Reflection.Binder and System.Reflection.BindingFlags classes.

An evidence parameter affects the security policy and permissions for the constructor. For more information, see the System.Security.Policy.Evidence class.

An instance of a type can be created at a local or remote site. If the type is created remotely, an activation attribute parameter specifies the URI of the remote site. The call to create the instance might pass through intermediary sites before it reaches the remote site. Other activation attributes can modify the environment, or context, in which the call operates at the remote and intermediary sites.

If the instance is created locally, a reference to that object is returned. If the instance is created remotely, a reference to a proxy is returned. The remote object is manipulated through the proxy as if it were a local object.

The GetObject method creates a proxy to a currently running remote object, server-activated well-known object, or XML Web service. You can specify the connection medium, that is, the channel. For more information, see the System.Runtime.Remoting.Channels.ChannelServices class.

Assemblies contain type definitions. The CreateInstance method creates an instance of a type from a currently running assembly. The CreateInstanceFrom method creates an instance from a file that contains an assembly. The CreateComInstanceFrom method creates an instance of a COM object from a file that contains an assembly.

Methods

Expand table

CreateInstance(String, String)	Creates an instance of the type whose name is specified, using the named assembly and parameterless constructor.
CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the type whose name is specified, using the named assembly and the constructor that best matches the specified parameters.
CreateInstance(String, String, Object[])	Creates an instance of the type whose name is specified, using the named assembly and parameterless constructor.
CreateInstance(Type)	Creates an instance of the specified type using that type's parameterless constructor.
CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the specified type using the constructor that best matches the specified

	parameters.
CreateInstance(Type, Boolean)	Creates an instance of the specified type using that type's parameterless constructor.
CreateInstance(Type, Object[])	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(Type, Object[], Object[])	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance<T>()	Creates an instance of the type designated by the specified generic type parameter, using the parameterless constructor.
CreateInstanceFrom(String, String)	Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.
CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the type whose name is specified, using the named assembly file and the constructor that best matches the specified parameters.
CreateInstanceFrom(String, String, Object[])	Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Activator.CreateInstance Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the specified type using the constructor that best matches the specified parameters.

Overloads

Expand table

CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the type whose name is specified, using the named assembly and the constructor that best matches the specified parameters.
CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(Type, Object[], Object[])	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(String, String, Object[])	Creates an instance of the type whose name is specified, using the named assembly and parameterless constructor.
CreateInstance(Type, Object[])	Creates an instance of the specified type using the constructor that best matches the specified parameters.
CreateInstance(Type, Boolean)	Creates an instance of the specified type using that type's parameterless constructor.
CreateInstance(String, String)	Creates an instance of the type whose name is specified, using the named assembly and

	parameterless constructor.
CreateInstance(Type)	Creates an instance of the specified type using that type's parameterless constructor.
CreateInstance<T>()	Creates an instance of the type designated by the specified generic type parameter, using the parameterless constructor.

CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates an instance of the type whose name is specified, using the named assembly and the constructor that best matches the specified parameters.

C#

```
public static System.Runtime.Remoting.ObjectHandle? CreateInstance
(string assemblyName, string typeName, bool ignoreCase,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, object?[]? args, System.Globalization.CultureInfo? culture,
object?[]? activationAttributes);
```

Parameters

`assemblyName` String

The name of the assembly where the type named `typeName` is sought. If `assemblyName` is `null`, the executing assembly is searched.

`typeName` String

The fully qualified name of the type to create an instance of.

`ignoreCase` Boolean

`true` to specify that the search for `typeName` is not case-sensitive; `false` to specify that the search is case-sensitive.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that uses `bindingAttr` and `args` to seek and identify the `typeName` constructor. If `binder` is `null`, the default binder is used.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `typeName` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created instance, or `null` for Nullable<T> instances with no value.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

TargetInvocationException

The constructor, which was invoked through reflection, threw an exception.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

NotSupportedException

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

-or-

The constructor that best matches `args` has `varargs` arguments.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

FileLoadException

An assembly or module was loaded twice with two different evidences.

-or-

The assembly name or code base is invalid.

Remarks

Use ObjectHandle.Unwrap to unwrap the return value.

> ⓘ **Note**
>
> Starting with .NET Framework 2.0, this method can be used to create nonpublic types and members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types and members is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates an instance of the specified type using the constructor that best matches the specified parameters.

C#

```
public static object? CreateInstance (Type type,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, object?[]? args, System.Globalization.CultureInfo? culture,
object?[]? activationAttributes);
```

Parameters

`type` Type

The type of object to create.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `type` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that uses `bindingAttr` and `args` to seek and identify the `type` constructor. If `binder` is `null`, the default binder is used.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `type` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances with no value.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

-or-

The constructor that best matches `args` has `varargs` arguments.

TargetInvocationException

The constructor being called throws an exception.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException
No matching constructor was found.

COMException
`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException
`type` is not a valid type.

Remarks

The constructor to be invoked must provide the most specific match with the specified argument list under the constraints of the specified binder and binding attributes.

> **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types and members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the nonpublic types and members is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)

Creates an instance of the specified type using the constructor that best matches the specified parameters.

C#

```
public static object? CreateInstance (Type type,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, object?[]? args, System.Globalization.CultureInfo? culture);
```

Parameters

`type` Type

The type of object to create.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `type` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that uses `bindingAttr` and `args` to seek and identify the `type` constructor. If `binder` is `null`, the default binder is used.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `type` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances with no value.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

-or-

The constructor that best matches `args` has `varargs` arguments.

TargetInvocationException

The constructor being called throws an exception.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException
No matching constructor was found.

COMException
`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException
`type` is not a valid type.

Remarks

The constructor to be invoked must provide the most specific match with the specified argument list under the constraints of the specified binder and binding attributes.

> **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types and members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types and members is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateInstance(Type, Object[], Object[])

Creates an instance of the specified type using the constructor that best matches the specified parameters.

```
C#

public static object? CreateInstance (Type type, object?[]? args, object?
[]? activationAttributes);
```

Parameters

`type` Type

The type of object to create.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances with no value.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

-or-

The constructor that best matches `args` has `varargs` arguments.

TargetInvocationException

The constructor being called throws an exception.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException

No matching public constructor was found.

COMException

`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException

`type` is not a valid type.

Remarks

The constructor to be invoked must be accessible and must provide the most specific match with the specified argument list.

> ⓘ **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateInstance(String, String, Object[])

Creates an instance of the type whose name is specified, using the named assembly and parameterless constructor.

```
C#
```

```
public static System.Runtime.Remoting.ObjectHandle? CreateInstance
(string assemblyName, string typeName, object?[]? activationAttributes);
```

Parameters

`assemblyName` String

The name of the assembly where the type named `typeName` is sought. If `assemblyName` is `null`, the executing assembly is searched.

`typeName` String

The fully qualified name of the type to create an instance of.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

NotSupportedException

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

-or-

`activationAttributes` is not a UrlAttribute

array.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

FileLoadException

An assembly or module was loaded twice with two different evidences.

-or-

The assembly name or code base is invalid.

TargetInvocationException

The invoked constructor throws an exception.

-or-

An error occurred when attempting remote activation in a target specified in `activationAttributes`.

Remarks

Use ObjectHandle.Unwrap to unwrap the return value.

> ⓘ **Note**
>
> Starting with .NET Framework 2.0, this method can be used to create nonpublic types if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the nonpublic types is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstance(Type, Object[])

Creates an instance of the specified type using the constructor that best matches the specified parameters.

C#

```
public static object? CreateInstance (Type type, params object?[]? args);
```

Parameters

`type` Type

The type of object to create.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances with no value.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

-or-

The constructor that best matches `args` has `varargs` arguments.

TargetInvocationException
The constructor being called throws an exception.

MethodAccessException
The caller does not have permission to call this constructor.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, MemberAccessException, instead.

MemberAccessException
Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException
The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException
No matching public constructor was found.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, MissingMemberException, instead.

COMException
`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException
`type` is not a valid type.

Examples

The following example calls the CreateInstance(Type, Object[]) method to create a String object. It calls the String.String(Char[], Int32, Int32) constructor to instantiate a string that contains ten elements from a character array starting at the fourteenth position.

```
C#

using System;
```

```
public class Example
{
   public static void Main()
   {
      // Initialize array of characters from a to z.
      char[] chars = new char[26];
      for (int ctr = 0; ctr < 26; ctr++)
         chars[ctr] = (char) (ctr + 0x0061);

      object obj = Activator.CreateInstance(typeof(string),
                                            new object[] { chars, 13, 10
} );
      Console.WriteLine(obj);
   }
}
// The example displays the following output:
//       nopqrstuvw
```

The following example creates a jagged array whose elements are arguments to be passed to a String constructor. The example then passes each array to the CreateInstance(Type, Object[]) method to invoke the appropriate string constructor.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      char[] characters = { 'a', 'b', 'c', 'd', 'e', 'f' };
      object[][] arguments = new object[3][] { new object[] { characters
},
                                               new object[] { characters,
1, 4 },
                                               new object[] {
characters[1], 20 } };

      for (int ctr = 0; ctr <= arguments.GetUpperBound(0); ctr++) {
         object[] args = arguments[ctr];
         object result = Activator.CreateInstance(typeof(string), args);
         Console.WriteLine("{0}: {1}", result.GetType().Name, result);
      }
   }
}
// The example displays the following output:
//    String: abcdef
//    String: bcde
//    String: bbbbbbbbbbbbbbbbbbb
```

Remarks

The constructor to be invoked must be accessible and must provide the most specific match with the specified argument list.

> ⓘ **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CreateInstance(Type, Boolean)

Creates an instance of the specified type using that type's parameterless constructor.

C#

```
public static object? CreateInstance (Type type, bool nonPublic);
```

Parameters

type Type

The type of object to create.

`nonPublic` Boolean

`true` if a public or nonpublic parameterless constructor can match; `false` if only a public parameterless constructor can match.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

TargetInvocationException

The constructor being called throws an exception.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException

No matching public constructor was found.

COMException

`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException

`type` is not a valid type.

Remarks

> **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types and members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types and members is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

CreateInstance(String, String)

Creates an instance of the type whose name is specified, using the named assembly and parameterless constructor.

```
C#

public static System.Runtime.Remoting.ObjectHandle? CreateInstance
(string assemblyName, string typeName);
```

Parameters

`assemblyName` String

The name of the assembly where the type named `typeName` is sought. If `assemblyName` is `null`, the executing assembly is searched.

`typeName` String

The fully qualified name of the type to create an instance of.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

You cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

TargetInvocationException

The constructor, which was invoked through reflection, threw an exception.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

NotSupportedException

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

FileLoadException

An assembly or module was loaded twice with two different evidences.

-or-

The assembly name or code base is invalid.

Examples

The following example defines a class named `Person` in an assembly named `PersonInfo`. Note that the `Person` class has two constructors, one of which is parameterless.

```
C#
```

```
using System;

public class Person
{
   private string _name;

   public Person()
   { }

   public Person(string name)
   {
      this._name = name;
   }

   public string Name
   { get { return this._name; }
     set { this._name = value; } }

   public override string ToString()
   {
      return this._name;
   }
}
```

The following example calls the CreateInstance(String, String) method to instantiate the `Person` class. It requires a reference to PersonInfo.dll to be added to the project. Because the CreateInstance(String, String) method calls the `Person` class parameterless constructor, the example assigns a value to its `Name` property.

C#

```
using System;
using System.Runtime.Remoting;

public class Example
{
   public static void Main()
   {
      ObjectHandle handle = Activator.CreateInstance("PersonInfo",
"Person");
      Person p = (Person) handle.Unwrap();
      p.Name = "Samuel";
      Console.WriteLine(p);
   }
}
// The example displays the following output:
//        Samuel
```

However, CreateInstance is frequently called to instantiate a type that crosses machine boundaries or that is not known at design time. In this case, you cannot include a reference to the assembly in the project and cannot make early-bound calls to the type's members. To work around this limitation, the following example uses the CreateInstance method along with reflection to assign a value to the `Person` object's `Name` property and to display its value.

C#

```
using System;
using System.Reflection;
using System.Runtime.Remoting;

public class Example
{
   public static void Main()
   {
      ObjectHandle handle = Activator.CreateInstance("PersonInfo",
"Person");
      object p = handle.Unwrap();
      Type t = p.GetType();
      PropertyInfo prop = t.GetProperty("Name");
      if (prop != null)
         prop.SetValue(p, "Samuel");

      MethodInfo method = t.GetMethod("ToString");
      object retVal = method.Invoke(p, null);
      if (retVal != null)
         Console.WriteLine(retVal);
   }
}
// The example displays the following output:
//        Samuel
```

Remarks

Use ObjectHandle.Unwrap to unwrap the return value.

`assemblyName` can be either of the following:

- The simple name of an assembly, without its path or file extension. For example, you would specify `TypeExtensions` for an assembly whose path and name are .\bin\TypeExtensions.dll.

- The full name of a signed assembly, which consists of its simple name, version, culture, and public key token; for example, "TypeExtensions, Version=1.0.0.0, Culture=neutral, PublicKeyToken=181869f2f7435b51".

For more information on how the common language runtime identifies and loads assemblies, see How the Runtime Locates Assemblies. For information on using the application configuration file to define assembly locations, see Specifying an Assembly's Location. If `assemblyName` is found, it is loaded in the default context.

> **Note**
>
> Starting with .NET Framework 2.0, this method can be used to create nonpublic types if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstance(Type)

Creates an instance of the specified type using that type's parameterless constructor.

C#

```
public static object? CreateInstance (Type type);
```

Parameters

`type` Type

The type of object to create.

Returns

Object

A reference to the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`type` is `null`.

ArgumentException

`type` is not a `RuntimeType`.

-or-

`type` is an open generic type (that is, the ContainsGenericParameters property returns `true`).

NotSupportedException

`type` cannot be a TypeBuilder.

-or-

Creation of TypedReference, ArgIterator, Void, and RuntimeArgumentHandle types, or arrays of those types, is not supported.

-or-

The assembly that contains `type` is a dynamic assembly that was created with Save.

TargetInvocationException

The constructor being called throws an exception.

MethodAccessException

The caller does not have permission to call this constructor.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, MemberAccessException, instead.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

InvalidComObjectException

The COM type was not obtained through GetTypeFromProgID or GetTypeFromCLSID.

MissingMethodException

No matching public constructor was found.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, MissingMemberException, instead.

COMException

`type` is a COM object but the class identifier used to obtain the type is invalid, or the identified class is not registered.

TypeLoadException

`type` is not a valid type.

Examples

The following code example demonstrates how to call the CreateInstance(Type) method. Instances of several different types are created and their default values are displayed.

C#

```
using System;

class DynamicInstanceList
{
    private static string instanceSpec =
"System.EventArgs;System.Random;" +
        "System.Exception;System.Object;System.Version";

    public static void Main()
    {
        string[] instances = instanceSpec.Split(';');
        Array instlist = Array.CreateInstance(typeof(object),
instances.Length);
        object item;
        for (int i = 0; i < instances.Length; i++)
        {
            // create the object from the specification string
            Console.WriteLine("Creating instance of: {0}", instances[i]);
            item = Activator.CreateInstance(Type.GetType(instances[i]));
            instlist.SetValue(item, i);
        }
        Console.WriteLine("\nObjects and their default values:\n");
        foreach (object o in instlist)
        {
            Console.WriteLine("Type:     {0}\nValue:    {1}\nHashCode:
{2}\n",
                o.GetType().FullName, o.ToString(), o.GetHashCode());
```

```
        }
    }
}

// This program will display output similar to the following:
//
// Creating instance of: System.EventArgs
// Creating instance of: System.Random
// Creating instance of: System.Exception
// Creating instance of: System.Object
// Creating instance of: System.Version
//
// Objects and their default values:
//
// Type:     System.EventArgs
// Value:    System.EventArgs
// HashCode: 46104728
//
// Type:     System.Random
// Value:    System.Random
// HashCode: 12289376
//
// Type:     System.Exception
// Value:    System.Exception: Exception of type 'System.Exception' was
thrown.
// HashCode: 55530882
//
// Type:     System.Object
// Value:    System.Object
// HashCode: 30015890
//
// Type:     System.Version
// Value:    0.0
// HashCode: 1048575
```

Remarks

The constructor to be invoked must be accessible.

> ⓘ **Note**
>
> Starting with .NET Framework 2.0, this method can be used to access nonpublic types if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the assembly that contains the nonpublic types is restricted to the caller's grant set or to a subset thereof. (See **Security Considerations for Reflection**.) To use this functionality, your application should target .NET Framework 3.5 or later.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CreateInstance<T>()

Creates an instance of the type designated by the specified generic type parameter, using the parameterless constructor.

C#

```
public static T CreateInstance<T> ();
```

Type Parameters

T

The type to create.

Returns

T

A reference to the newly created object, or `null` for Nullable<T> instances.

Exceptions

MissingMethodException

Cannot create an instance of an abstract class, or the type that is specified for `T` does not have a parameterless constructor.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, MissingMemberException, instead.

Remarks

The CreateInstance<T>() generic method is used by compilers to implement the instantiation of types specified by type parameters. For example, in the following generic method, the implementation of `new T()` (`gcnew T()` in C++) uses the CreateInstance<T>() generic method.

```
C#

public static T Factory<T>() where T : new()
{
    return new T();
}
```

In general, there is no use for the CreateInstance<T>() generic method in application code, because the type must be known at compile time. If the type is known at compile time, normal instantiation syntax can be used (`new` operator in C#, `New` in Visual Basic, `gcnew` in C++). If the type is not known at compile time, you can call a non-generic overload of CreateInstance.

There are no overloads of the CreateInstance<T>() generic method that take argument lists, because the non-generic overloads of CreateInstance already provide late-bound constructor resolution.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Activator.CreateInstanceFrom Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the type whose name is specified, using the named assembly file and the constructor that best matches the specified parameters.

Overloads

Expand table

CreateInstanceFrom(String, String, Object[])	Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.
CreateInstanceFrom(String, String)	Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.
CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates an instance of the type whose name is specified, using the named assembly file and the constructor that best matches the specified parameters.

CreateInstanceFrom(String, String, Object[])

Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.

C#

```
public static System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom
(string assemblyFile, string typeName, object?[]? activationAttributes);
```

Parameters

`assemblyFile` String

The name of a file that contains an assembly where the type named `typeName` is sought.

`typeName` String

The name of the type to create an instance of.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyFile`.

FileNotFoundException

`assemblyFile` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

TargetInvocationException

The constructor, which was invoked through reflection, threw an exception.

NotSupportedException

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

SecurityException

The caller does have the required FileIOPermission.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

Remarks

Use the ObjectHandle.Unwrap method to unwrap the return value.

For information about other exceptions that can be thrown by invoked methods, see the Exceptions section of the Assembly.LoadFrom and CreateInstance methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFrom(String, String)

Creates an instance of the type whose name is specified, using the named assembly file and parameterless constructor.

```
C#

public static System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom
(string assemblyFile, string typeName);
```

Parameters

`assemblyFile` String

The name of a file that contains an assembly where the type named `typeName` is sought.

`typeName` String

The name of the type to create an instance of.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyFile`.

FileNotFoundException

`assemblyFile` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

TargetInvocationException

The constructor, which was invoked through reflection, threw an exception.

SecurityException

The caller does have the required FileIOPermission.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

Examples

The following code example demonstrates how to call the CreateInstanceFrom method. This code example is part of a larger example provided for the Activator class.

C#

```
// Create an instance of the SomeType class that is defined in this
// assembly.
System.Runtime.Remoting.ObjectHandle oh =
    Activator.CreateInstanceFrom(Assembly.GetEntryAssembly().CodeBase,
                                 typeof(SomeType).FullName);

// Call an instance method defined by the SomeType type using this
object.
SomeType st = (SomeType) oh.Unwrap();

st.DoSomething(5);
```

Remarks

Use the ObjectHandle.Unwrap method to unwrap the return value.

For information about other exceptions that can be thrown by invoked methods, see the Exceptions section of the Assembly.LoadFrom and CreateInstance methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates an instance of the type whose name is specified, using the named assembly file and the constructor that best matches the specified parameters.

C#

```
public static System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom
(string assemblyFile, string typeName, bool ignoreCase,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, object?[]? args, System.Globalization.CultureInfo? culture,
object?[]? activationAttributes);
```

Parameters

`assemblyFile` String

The name of a file that contains an assembly where the type named `typeName` is sought.

`typeName` String

The name of the type to create an instance of.

`ignoreCase` Boolean

`true` to specify that the search for `typeName` is not case-sensitive; `false` to specify that the search is case-sensitive.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that uses `bindingAttr` and `args` to seek and identify the `typeName` constructor. If `binder` is `null`, the default binder is used.

`args` Object[]

An array of arguments that match in number, order, and type the parameters of the constructor to invoke. If `args` is an empty array or `null`, the constructor that takes no parameters (the parameterless constructor) is invoked.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `typeName` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. This is typically an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

A handle that must be unwrapped to access the newly created object, or `null` for Nullable<T> instances.

Exceptions

ArgumentNullException

`typeName` is `null`.

MissingMethodException

No matching constructor was found.

TypeLoadException

`typename` was not found in `assemblyFile`.

FileNotFoundException

`assemblyFile` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

TargetInvocationException

The constructor, which was invoked through reflection, threw an exception.

SecurityException

The caller does not have the required FileIOPermission.

NotSupportedException

`activationAttributes` is not an empty array, and the type being created does not derive from MarshalByRefObject.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

The common language runtime (CLR) version 2.0 or later is currently loaded, and `assemblyName` was compiled for a version of the CLR that is later than the currently loaded version. Note that the .NET Framework versions 2.0, 3.0, and 3.5 all use CLR version 2.0.

Remarks

Use the ObjectHandle.Unwrap method to unwrap the return value.

For information about other exceptions that can be thrown by invoked methods, see the Exceptions section of the Assembly.LoadFrom and CreateInstance methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents one or more errors that occur during application execution.

C#

```
public class AggregateException : Exception
```

Inheritance Object → Exception → AggregateException

Examples

The following example catches the AggregateException exception and calls the Handle method to handle each exception it contains. Compiling and running the example with the first `task1` variable should result in an AggregateException object that contains an UnauthorizedAccessException exception. Commenting out that line, uncommenting the second `task1` variable, and compiling and running the example produces an AggregateException object that contains an IndexOutOfRangeException exception.

C#

```
using System;
using System.IO;
using System.Threading.Tasks;

class Example
{
   static async Task Main(string[] args)
   {
      // Get a folder path whose directories should throw an
UnauthorizedAccessException.
      string path = Directory.GetParent(
                                Environment.GetFolderPath(

Environment.SpecialFolder.UserProfile)).FullName;

      // Use this line to throw UnauthorizedAccessException, which we
```

```
handle.
      Task<string[]> task1 = Task<string[]>.Factory.StartNew(() =>
GetAllFiles(path));

      // Use this line to throw an exception that is not handled.
      // Task task1 = Task.Factory.StartNew(() => { throw new
IndexOutOfRangeException(); } );
      try
      {
          await task1;
      }
      catch (UnauthorizedAccessException ae)
      {
          Console.WriteLine("Caught unauthorized access exception-await
behavior");
      }
      catch (AggregateException ae)
      {
          Console.WriteLine("Caught aggregate exception-Task.Wait
behavior");
          ae.Handle((x) =>
          {
              if (x is UnauthorizedAccessException) // This we know how to
handle.
              {
                  Console.WriteLine("You do not have permission to access
all folders in this path.");
                  Console.WriteLine("See your network administrator or try
another path.");
                  return true;
              }
              return false; // Let anything else stop the application.
          });
      }

      Console.WriteLine("task1 Status: {0}{1}", task1.IsCompleted ?
"Completed," : "",
                                                  task1.Status);
   }

   static string[] GetAllFiles(string str)
   {
      // Should throw an UnauthorizedAccessException exception.
      return System.IO.Directory.GetFiles(str, "*.txt",
System.IO.SearchOption.AllDirectories);
   }
}
// The example displays the following output if the file access task is run:
//       You do not have permission to access all folders in this path.
//       See your network administrator or try another path.
//       task1 Status: Completed,Faulted
// It displays the following output if the second task is run:
//       Unhandled Exception: System.AggregateException: One or more errors
occurred. ---
//       > System.IndexOutOfRangeException: Index was outside the bounds of
```

```
the array.
//          at Example.<Main>b__0()
//          at System.Threading.Tasks.Task.Execute()
//          --- End of inner exception stack trace ---
//          at System.AggregateException.Handle(Func`2 predicate)
//          at Example.Main(String[] args)
```

Remarks

AggregateException is used to consolidate multiple failures into a single, throwable exception object. It is used extensively in the Task Parallel Library (TPL) and Parallel LINQ (PLINQ). For more information, see Exception Handling and How to: Handle Exceptions in a PLINQ Query. For additional information, see the Aggregating Exceptions entry in the .NET Matters blog.

Constructors

Expand table

AggregateException()	Initializes a new instance of the AggregateException class with a system-supplied message that describes the error.
AggregateException(Exception[])	Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.
Aggregate Exception(IEnumerable<Exception>)	Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.
AggregateException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AggregateException class with serialized data.
AggregateException(String)	Initializes a new instance of the AggregateException class with a specified message that describes the error.
AggregateException(String, Exception)	Initializes a new instance of the AggregateException class with a specified error message and a reference to the inner exception that is the cause of this exception.
AggregateException(String, Exception[])	Initializes a new instance of the AggregateException class with a specified error message and references

	to the inner exceptions that are the cause of this exception.
AggregateException(String, IEnumerable<Exception>)	Initializes a new instance of the AggregateException class with a specified error message and references to the inner exceptions that are the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Inner Exceptions	Gets a read-only collection of the Exception instances that caused the current exception.
Message	Gets a message that describes the exception.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.

	(Inherited from Object)
Flatten()	Flattens an AggregateException instances into a single, new instance.
GetBaseException()	Returns the AggregateException that is the root cause of this exception.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AggregateException class with serialized data.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
Handle(Func<Exception,Boolean>)	Invokes a handler on each Exception contained by this AggregateException.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current AggregateException.

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Thread Safety

All public and protected members of AggregateException are thread-safe and may be used concurrently from multiple threads.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the AggregateException class.

Overloads

Expand table

AggregateException()	Initializes a new instance of the AggregateException class with a system-supplied message that describes the error.
AggregateException(IEnumerable<Exception>)	Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.
AggregateException(Exception[])	Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.
AggregateException(String)	Initializes a new instance of the AggregateException class with a specified message that describes the error.
AggregateException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AggregateException class with serialized data.
AggregateException(String, IEnumerable<Exception>)	Initializes a new instance of the AggregateException class with a specified error message and references to the inner exceptions that are the cause of this exception.
AggregateException(String, Exception)	Initializes a new instance of the AggregateException class with a specified

	error message and a reference to the inner exception that is the cause of this exception.
AggregateException(String, Exception[])	Initializes a new instance of the AggregateException class with a specified error message and references to the inner exceptions that are the cause of this exception.

AggregateException()

Initializes a new instance of the AggregateException class with a system-supplied message that describes the error.

```
C#

public AggregateException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "System.AggregateException: One or more errors occurred." This message takes into account the current system culture.

The following table shows the initial property values for an instance of AggregateException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(IEnumerable<Exception>)

Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.

C#

```
public AggregateException
(System.Collections.Generic.IEnumerable<Exception> innerExceptions);
```

Parameters

`innerExceptions` IEnumerable<Exception>

The exceptions that are the cause of the current exception.

Exceptions

ArgumentNullException

The `innerExceptions` argument is null.

ArgumentException

An element of `innerExceptions` is null.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(Exception[])

Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.

C#

```
public AggregateException (params Exception[] innerExceptions);
```

Parameters

`innerExceptions` Exception[]

The exceptions that are the cause of the current exception.

Exceptions

ArgumentNullException

The `innerExceptions` argument is null.

ArgumentException

An element of `innerExceptions` is null.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(String)

Initializes a new instance of the AggregateException class with a specified message that describes the error.

C#

```
public AggregateException (string? message);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of AggregateException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the AggregateException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected AggregateException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` argument is null.

SerializationException

The exception could not be deserialized correctly.

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AggregateException(String, IEnumerable<Exception>)

Initializes a new instance of the AggregateException class with a specified error message and references to the inner exceptions that are the cause of this exception.

C#

```
public AggregateException (string? message,
System.Collections.Generic.IEnumerable<Exception> innerExceptions);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerExceptions` IEnumerable<Exception>

The exceptions that are the cause of the current exception.

Exceptions

ArgumentNullException

The `innerExceptions` argument is null.

ArgumentException

An element of `innerExceptions` is null.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(String, Exception)

Initializes a new instance of the AggregateException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public AggregateException (string? message, Exception innerException);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Exceptions

ArgumentNullException

The `innerException` argument is null.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of AggregateException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

AggregateException(String, Exception[])

Initializes a new instance of the AggregateException class with a specified error message and references to the inner exceptions that are the cause of this exception.

C#

```
public AggregateException (string? message, params Exception[]
innerExceptions);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerExceptions` Exception[]

The exceptions that are the cause of the current exception.

Exceptions

ArgumentNullException

The `innerExceptions` argument is null.

ArgumentException

An element of `innerExceptions` is null.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AggregateException.InnerExceptions Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a read-only collection of the Exception instances that caused the current exception.

C#

```
public System.Collections.ObjectModel.ReadOnlyCollection<Exception>
InnerExceptions { get; }
```

Property Value

ReadOnlyCollection<Exception>

A read-only collection of the Exception instances that caused the current exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

AggregateException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a message that describes the exception.

C#

```
public override string Message { get; }
```

Property Value

String

The message that describes the exception.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException.Flatten Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Flattens an AggregateException instances into a single, new instance.

C#

```
public AggregateException Flatten ();
```

Returns

AggregateException

A new, flattened AggregateException.

Examples

In the following example, nested AggregateException instances are flattened and handled in just one loop.

C#

```
using System;
using System.Threading.Tasks;

public class Example
{
   public static void Main()
   {
      var task1 = Task.Factory.StartNew(() => {
                      var child1 = Task.Factory.StartNew(() => {
                         var child2 = Task.Factory.StartNew(() => {
                             // This exception is nested inside three
AggregateExceptions.
                             throw new CustomException("Attached child2
faulted.");
                         }, TaskCreationOptions.AttachedToParent);

                         // This exception is nested inside two
```

```
AggregateExceptions.
                        throw new CustomException("Attached child1
faulted.");
                    }, TaskCreationOptions.AttachedToParent);
      });

      try {
         task1.Wait();
      }
      catch (AggregateException ae) {
         foreach (var e in ae.Flatten().InnerExceptions) {
            if (e is CustomException) {
               Console.WriteLine(e.Message);
            }
            else {
               throw;
            }
         }
      }
   }
}

public class CustomException : Exception
{
   public CustomException(String message) : base(message)
   {}
}
// The example displays the following output:
//    Attached child1 faulted.
//    Attached child2 faulted.
```

You can also use the AggregateException.Flatten method to rethrow the inner exceptions from multiple AggregateException instances thrown by multiple tasks in a single AggregateException instance, as the following example shows.

C#

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;

public class Example2
{
   public static void Main()
   {
        try {
            ExecuteTasks();
        }
        catch (AggregateException ae) {
            foreach (var e in ae.InnerExceptions) {
                Console.WriteLine("{0}:\n   {1}", e.GetType().Name,
```

```
e.Message);
            }
        }
    }

    static void ExecuteTasks()
    {
        // Assume this is a user-entered String.
        String path = @"C:\";
        List<Task> tasks = new List<Task>();

        tasks.Add(Task.Run(() => {
                                // This should throw an
UnauthorizedAccessException.
                                return Directory.GetFiles(path, "*.txt",

SearchOption.AllDirectories);
                            }));

        tasks.Add(Task.Run(() => {
                                if (path == @"C:\")
                                   throw new ArgumentException("The system
root is not a valid path.");
                                return new String[] { ".txt", ".dll", ".exe",
".bin", ".dat" };
                            }));

        tasks.Add(Task.Run(() => {
                                 throw new NotImplementedException("This
operation has not been implemented.");
                            }));

        try {
            Task.WaitAll(tasks.ToArray());
        }
        catch (AggregateException ae) {
            throw ae.Flatten();
        }
    }
}
// The example displays the following output:
//       UnauthorizedAccessException:
//          Access to the path 'C:\Documents and Settings' is denied.
//       ArgumentException:
//          The system root is not a valid path.
//       NotImplementedException:
//          This operation has not been implemented.
```

Remarks

If a task has an attached child task that throws an exception, that exception is wrapped in an AggregateException exception before it is propagated to the parent task, which

wraps that exception in its own AggregateException exception before it propagates it back to the calling thread. In such cases, the InnerExceptions property of the AggregateException exception that is caught by the Task.Wait, Wait, Task.WaitAny or Task.WaitAll method contains one or more AggregateException instances, not the original exceptions that caused the fault. To avoid having to iterate over nested AggregateException exceptions, you can use the Flatten method to remove all the nested AggregateException exceptions, so that the InnerExceptions property of the returned AggregateException object contains the original exceptions.

This method recursively flattens all instances of AggregateException exceptions that are inner exceptions of the current AggregateException instance. The inner exceptions returned in the new AggregateException are the union of all the inner exceptions from exception tree rooted at the current AggregateException instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception Handling (Task Parallel Library)
- Aggregating Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException.GetBaseException Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the AggregateException that is the root cause of this exception.

C#

```
public override Exception GetBaseException ();
```

Returns

Exception

The AggregateException that is the root cause of this exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

AggregateException.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the AggregateException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` argument is null.

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- XML and SOAP Serialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException.Handle(Func<Exception,Boolean>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Invokes a handler on each Exception contained by this AggregateException.

```
C#
public void Handle (Func<Exception,bool> predicate);
```

Parameters

`predicate` Func<Exception,Boolean>

The predicate to execute for each exception. The predicate accepts as an argument the Exception to be processed and returns a Boolean to indicate whether the exception was handled.

Exceptions

ArgumentNullException

The `predicate` argument is null.

AggregateException

An exception contained by this AggregateException was not handled.

Examples

Ordinarily, an exception handler that catches an AggregateException exception uses a `foreach` loop (in C#) or `For Each` loop (in Visual Basic) to handle each exception in its InnerExceptions collection. Instead, the following example uses the Handle method to

handle each exception, and only rethrows exceptions that are not `CustomException` instances.

C#

```
using System;
using System.Threading.Tasks;

public class Example
{
   public static void Main()
   {
      var task1 = Task.Run( () => { throw new CustomException("This
exception is expected!"); } );

      try {
          task1.Wait();
      }
      catch (AggregateException ae)
      {
         // Call the Handle method to handle the custom exception,
         // otherwise rethrow the exception.
         ae.Handle(ex => { if (ex is CustomException)
                              Console.WriteLine(ex.Message);
                            return ex is CustomException;
                          });
      }
   }
}

public class CustomException : Exception
{
   public CustomException(String message) : base(message)
   {}
}
// The example displays the following output:
//        This exception is expected!
```

The following is a more complete example that uses the Handle method to provide special handling for an UnauthorizedAccessException when enumerating files.

C#

```
using System;
using System.IO;
using System.Threading.Tasks;

public class Example2
{
    public static void Main()
    {
        // This should throw an UnauthorizedAccessException.
```

```
        try {
            var files = GetAllFiles(@"C:\");
            if (files != null)
               foreach (var file in files)
                  Console.WriteLine(file);
         }
         catch (AggregateException ae) {
            foreach (var ex in ae.InnerExceptions)
                Console.WriteLine("{0}: {1}", ex.GetType().Name, ex.Message);
         }
         Console.WriteLine();

         // This should throw an ArgumentException.
         try {
            foreach (var s in GetAllFiles(""))
               Console.WriteLine(s);
         }
         catch (AggregateException ae) {
            foreach (var ex in ae.InnerExceptions)
                Console.WriteLine("{0}: {1}", ex.GetType().Name, ex.Message);
         }
     }

     static string[] GetAllFiles(string path)
     {
        var task1 = Task.Run( () => Directory.GetFiles(path, "*.txt",

SearchOption.AllDirectories));

        try {
           return task1.Result;
        }
        catch (AggregateException ae) {
           ae.Handle( x => { // Handle an UnauthorizedAccessException
                             if (x is UnauthorizedAccessException) {
                                 Console.WriteLine("You do not have
permission to access all folders in this path.");
                                 Console.WriteLine("See your network
administrator or try another path.");
                             }
                             return x is UnauthorizedAccessException;
                          });
           return Array.Empty<String>();
        }
    }
}
// The example displays the following output:
//       You do not have permission to access all folders in this path.
//       See your network administrator or try another path.
//
//       ArgumentException: The path is not of a legal form.
```

Remarks

Each invocation of the `predicate` returns true or false to indicate whether the Exception was handled. After all invocations, if any exceptions went unhandled, all unhandled exceptions will be put into a new AggregateException which will be thrown. Otherwise, the Handle method simply returns. If any invocations of the `predicate` throws an exception, it will halt the processing of any more exceptions and immediately propagate the thrown exception as-is.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AggregateException.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates and returns a string representation of the current AggregateException.

C#

```
public override string ToString ();
```

Returns

String

A string representation of the current exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

AppContext Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides members for setting and retrieving data about an application's context.

C#

```
public static class AppContext
```

Inheritance Object → AppContext

Remarks

For more information about this API, see Supplemental API remarks for AppContext.

Properties

Expand table

BaseDirectory	Gets the file path of the base directory that the assembly resolver uses to probe for assemblies.
TargetFrameworkName	Gets the name of the framework version targeted by the current application.

Methods

Expand table

GetData(String)	Returns the value of the named data element assigned to the current application domain.
SetData(String, Object)	Sets the value of the named data element assigned to the current application domain.

SetSwitch(String, Boolean)	Sets the value of a switch.
TryGetSwitch(String, Boolean)	Tries to get the value of a switch.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- <runtime> Element
- <AppContextSwitchOverrides> Element

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppContext.BaseDirectory Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the file path of the base directory that the assembly resolver uses to probe for assemblies.

C#

```
public static string BaseDirectory { get; }
```

Property Value

String

The file path of the base directory that the assembly resolver uses to probe for assemblies.

Remarks

This is a per-application domain property. Its value corresponds to the AppDomain.BaseDirectory property of the current application domain.

In .NET 5 and later versions, for bundled assemblies, the value returned is the containing directory of the host executable.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppContext.TargetFrameworkName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the framework version targeted by the current application.

C#

```
public static string? TargetFrameworkName { get; }
```

Property Value

String

The name of the framework version targeted by the current application.

Remarks

The name of the target framework version corresponds to the value of the TargetFrameworkAttribute.FrameworkName property. For a list of target framework names for .NET Framework, see the <supportedRuntime> Element element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.6, 2.0, 2.1

See also

- TargetFrameworkAttribute
- FrameworkName

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppContext.GetData(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the value of the named data element assigned to the current application domain.

C#

```
public static object? GetData (string name);
```

Parameters

name String

The name of the data element.

Returns

Object

The value of `name`, if `name` identifies a named value; otherwise, `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppContext.SetData(String, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the value of the named data element assigned to the current application domain.

C#

```
public static void SetData (string name, object? data);
```

Parameters

name String

The name of the data element

data Object

The value of `name`

Exceptions

ArgumentNullException

`name` is `null`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppContext.SetSwitch(String, Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the value of a switch.

```
C#
public static void SetSwitch (string switchName, bool isEnabled);
```

Parameters

`switchName` String

The name of the switch.

`isEnabled` Boolean

The value of the switch.

Exceptions

ArgumentNullException

`switchName` is `null`.

ArgumentException

`switchName` is Empty.

Examples

The following line of code sets a switch named `Switch.AmazingLib.ThrowOnException` to `true`, which enables a legacy behavior. The library can then check whether a library consumer has set the value of the switch by calling the TryGetSwitch method.

```
C#
```

```
AppContext.SetSwitch("Switch.AmazingLib.ThrowOnException", true);
```

Remarks

The AppContext class enables library writers to provide a uniform opt-out mechanism for new functionality for their users. It establishes a loosely coupled contract between components in order to communicate an opt-out request. This capability is typically important when a change is made to existing functionality. Conversely, there is already an implicit opt-in for new functionality.

The SetSwitch method is called by an application (or a library) to declare the value of a switch (which is always a Boolean value) that a dependent library defines. The switch is always implicitly `false`, which provides the new behavior. Setting the switch to `true` enables it, which provides the legacy behavior. Explicitly setting the switch to `false` also provides the new behavior. The dependent library can then check the value of the switch by calling the TryGetSwitch method.

> **Note**
>
> It's beneficial to use a consistent format for switch names, since they are a formal contract exposed by a library. The following are two obvious formats.
>
> - *Switch.namespace.switchname*
> - *Switch.library.switchname*

For applications running on .NET Framework, in addition to setting the value of a switch programmatically, it can also be set:

- By adding the switch name and value to the <AppContextSwitchOverrides> element in the <runtime> section of an application configuration file. For example, the following defines a switch named `Libraries.FPLibrary.UseExactFloatingPointComparison` whose value is `False`.

  XML

  ```
  <configuration>
     <runtime>
        <AppContextSwitchOverrides
  value="Libraries.FPLibrary.UseExactFloatingPointComparison=false" />
     </runtime>
  </configuration>
  ```

- By adding a string value whose name is the name of the switch to the **HKLM\SOFTWARE\Microsoft\.NETFramework\AppContext** (and **HKLM\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\AppContext**) subkeys in the registry. Its value must be the string representation of a Boolean that can be parsed by the Boolean.Parse method; that is, it must be "True", "true", "False", or "false".

If the registry entry exists, its value is overwritten by the `isEnabled` argument when SetSwitch is called. That is, the most recent call to the SetSwitch method overrides the value defined in the registry, in an app configuration file, or by previous calls to the SetSwitch method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- TryGetSwitch(String, Boolean)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppContext.TryGetSwitch(String, Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to get the value of a switch.

C#

```
public static bool TryGetSwitch (string switchName, out bool isEnabled);
```

Parameters

`switchName` String

The name of the switch.

`isEnabled` Boolean

When this method returns, contains the value of `switchName` if `switchName` was found, or `false` if `switchName` was not found. This parameter is passed uninitialized.

Returns

Boolean

`true` if `switchName` was set and the `isEnabled` argument contains the value of the switch; otherwise, `false`.

Exceptions

ArgumentNullException

`switchName` is `null`.

ArgumentException

`switchName` is Empty.

Examples

The following example determines whether a library consumer has set a switch named `Switch.AmazingLib.ThrowOnException`.

C#

```
public class AmazingLib
{
   private bool shouldThrow;

   public void PerformAnOperation()
   {
      if (!AppContext.TryGetSwitch("Switch.AmazingLib.ThrowOnException", out
shouldThrow)) {
         // This is the case where the switch value was not set by the
application.
         // The library can choose to get the value of shouldThrow by other
means.
         // If no overrides or default values are specified, the value
should be 'false'.
         // A false value implies the latest behavior.
      }

      // The library can use the value of shouldThrow to throw exceptions or
not.
      if (shouldThrow) {
         // old code
      }
      else {
          // new code
      }
   }
}
```

Remarks

The AppContext class enables library writers to provide a uniform opt-out mechanism for new functionality for their users. It establishes a loosely coupled contract between components in order to communicate an opt-out request. This capability is typically important when a change is made to existing functionality. Conversely, there is already an implicit opt-in for new functionality.

The common language runtime automatically populates the switches assigned to an AppContext instance by reading the registry and the application's configuration file. The value of these switches can then be overridden, and new switches added, by calling the SetSwitch method.

A library calls the TryGetSwitch method to check whether its consumers have declared the value of the switch and then act appropriately on it. By default, if the switch is not defined, the new functionality is enabled. If the switch is defined and its value is `false`, the new functionality is also enabled. If its value is `true`, the legacy behavior is enabled.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- SetSwitch(String, Boolean)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an application domain, which is an isolated environment where applications execute. This class cannot be inherited.

C#

```
public sealed class AppDomain : MarshalByRefObject
```

Inheritance Object → MarshalByRefObject → AppDomain

Examples

This example shows how to create a new AppDomain, instantiate a type in that new AppDomain, and communicate with that type's object. In addition, this example shows how to unload the AppDomain causing the object to be garbage collected.

C#

```
using System;
using System.Reflection;
using System.Threading;

class Module1
{
    public static void Main()
    {
        // Get and display the friendly name of the default AppDomain.
        string callingDomainName = Thread.GetDomain().FriendlyName;
        Console.WriteLine(callingDomainName);

        // Get and display the full name of the EXE assembly.
        string exeAssembly = Assembly.GetEntryAssembly().FullName;
        Console.WriteLine(exeAssembly);

        // Construct and initialize settings for a second AppDomain.
        AppDomainSetup ads = new AppDomainSetup();
        ads.ApplicationBase = AppDomain.CurrentDomain.BaseDirectory;
```

```
        ads.DisallowBindingRedirects = false;
        ads.DisallowCodeDownload = true;
        ads.ConfigurationFile =
            AppDomain.CurrentDomain.SetupInformation.ConfigurationFile;

        // Create the second AppDomain.
        AppDomain ad2 = AppDomain.CreateDomain("AD #2", null, ads);

        // Create an instance of MarshalbyRefType in the second AppDomain.
        // A proxy to the object is returned.
        MarshalByRefType mbrt =
            (MarshalByRefType) ad2.CreateInstanceAndUnwrap(
                exeAssembly,
                typeof(MarshalByRefType).FullName
            );

        // Call a method on the object via the proxy, passing the
        // default AppDomain's friendly name in as a parameter.
        mbrt.SomeMethod(callingDomainName);

        // Unload the second AppDomain. This deletes its object and
        // invalidates the proxy object.
        AppDomain.Unload(ad2);
        try
        {
            // Call the method again. Note that this time it fails
            // because the second AppDomain was unloaded.
            mbrt.SomeMethod(callingDomainName);
            Console.WriteLine("Sucessful call.");
        }
        catch(AppDomainUnloadedException)
        {
            Console.WriteLine("Failed call; this is expected.");
        }
    }
}

// Because this class is derived from MarshalByRefObject, a proxy
// to a MarshalByRefType object can be returned across an AppDomain
// boundary.
public class MarshalByRefType : MarshalByRefObject
{
    //  Call this method via a proxy.
    public void SomeMethod(string callingDomainName)
    {
        // Get this AppDomain's settings and display some of them.
        AppDomainSetup ads = AppDomain.CurrentDomain.SetupInformation;
        Console.WriteLine("AppName={0}, AppBase={1}, ConfigFile={2}",
            ads.ApplicationName,
            ads.ApplicationBase,
            ads.ConfigurationFile
        );

        // Display the name of the calling AppDomain and the name
```

```
        // of the second domain.
        // NOTE: The application's thread has transitioned between
        // AppDomains.
        Console.WriteLine("Calling from '{0}' to '{1}'.",
            callingDomainName,
            Thread.GetDomain().FriendlyName
        );
    }
}

/* This code produces output similar to the following:

AppDomainX.exe
AppDomainX, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
AppName=, AppBase=C:\AppDomain\bin,
ConfigFile=C:\AppDomain\bin\AppDomainX.exe.config
Calling from 'AppDomainX.exe' to 'AD #2'.
Failed call; this is expected.
 */
```

Remarks

Application domains, which are represented by AppDomain objects, help provide isolation, unloading, and security boundaries for executing managed code.

- Use application domains to isolate tasks that might bring down a process. If the state of the AppDomain that's executing a task becomes unstable, the AppDomain can be unloaded without affecting the process. This is important when a process must run for long periods without restarting. You can also use application domains to isolate tasks that should not share data.

- If an assembly is loaded into the default application domain, it cannot be unloaded from memory while the process is running. However, if you open a second application domain to load and execute the assembly, the assembly is unloaded when that application domain is unloaded. Use this technique to minimize the working set of long-running processes that occasionally use large DLLs.

> **Note**
>
> On .NET Core, the **AppDomain** implementation is limited by design and does not provide isolation, unloading, or security boundaries. For .NET Core, there is exactly one **AppDomain**. Isolation and unloading are provided through **AssemblyLoadContext**. Security boundaries should be provided by process boundaries and appropriate remoting techniques.

Multiple application domains can run in a single process; however, there is not a one-to-one correlation between application domains and threads. Several threads can belong to a single application domain, and while a given thread is not confined to a single application domain, at any given time, a thread executes in a single application domain.

Application domains are created using the CreateDomain method. AppDomain instances are used to load and execute assemblies (Assembly). When an AppDomain is no longer in use, it can be unloaded.

The AppDomain class implements a set of events that enable applications to respond when an assembly is loaded, when an application domain will be unloaded, or when an unhandled exception is thrown.

For more information on using application domains, see Application Domains.

This class implements the MarshalByRefObject, _AppDomain, and IEvidenceFactory interfaces.

You should never create a remotable wrapper for an AppDomain object. Doing so could publish a remote reference to that AppDomain, exposing methods such as CreateInstance to remote access and effectively destroying code access security for that AppDomain. Malicious clients connecting to the remoted AppDomain could obtain access to any resource the AppDomain itself has access to. Do not create remotable wrappers for any type that extends MarshalByRefObject and that implements methods that could be used by malicious clients to bypass the security system.

> **Caution**
>
> The default value for the **AppDomainSetup.DisallowCodeDownload** property is `false`. This setting is unsafe for services. To prevent services from downloading partially trusted code, set this property to `true`.

Properties

Expand table

BaseDirectory	Gets the base directory that the assembly resolver uses to probe for assemblies.
CurrentDomain	Gets the current application domain for the current Thread.
DynamicDirectory	Gets the directory that the assembly resolver uses to probe for

	dynamically created assemblies.
FriendlyName	Gets the friendly name of this application domain.
Id	Gets an integer that uniquely identifies the application domain within the process.
IsFullyTrusted	Gets a value that indicates whether assemblies that are loaded into the current application domain execute with full trust.
IsHomogenous	Gets a value that indicates whether the current application domain has a set of permissions that is granted to all assemblies that are loaded into the application domain.
MonitoringIsEnabled	Gets or sets a value that indicates whether CPU and memory monitoring of application domains is enabled for the current process. Once monitoring is enabled for a process, it cannot be disabled.
MonitoringSurvived MemorySize	Gets the number of bytes that survived the last collection and that are known to be referenced by the current application domain.
MonitoringSurvived ProcessMemorySize	Gets the total bytes that survived from the last collection for all application domains in the process.
MonitoringTotal AllocatedMemorySize	Gets the total size, in bytes, of all memory allocations that have been made by the application domain since it was created, without subtracting memory that has been collected.
MonitoringTotal ProcessorTime	Gets the total processor time that has been used by all threads while executing in the current application domain, since the process started.
PermissionSet	**Obsolete.** Gets the permission set of a sandboxed application domain.
RelativeSearchPath	Gets the path under the base directory where the assembly resolver should probe for private assemblies.
SetupInformation	Gets the application domain configuration information for this instance.
ShadowCopyFiles	Gets an indication whether the application domain is configured to shadow copy files.

Methods

Expand table

AppendPrivatePath(String)	**Obsolete.**

	Appends the specified directory name to the private path list.
ApplyPolicy(String)	Returns the assembly display name after policy has been applied.
ClearPrivatePath()	**Obsolete.** Resets the path that specifies the location of private assemblies to the empty string ("").
ClearShadowCopyPath()	**Obsolete.** Resets the list of directories containing shadow copied assemblies to the empty string ("").
CreateDomain(String)	**Obsolete.** Creates a new application domain with the specified name.
CreateInstance(String, String)	Creates a new instance of the specified type defined in the specified assembly.
CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly. Parameters specify a binder, binding flags, constructor arguments, culture-specific information used to interpret arguments, and optional activation attributes.
CreateInstance(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly. A parameter specifies an array of activation attributes.
CreateInstanceAndUnwrap(String, String)	Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, and the name of the type.
CreateInstanceAndUnwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.
CreateInstanceAndUnwrap(String, String, Object[])	Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, the name of the type, and an array of activation attributes.
CreateInstanceFrom(String, String)	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder,	Creates a new instance of the specified type defined in the specified assembly file.

Object[], CultureInfo, Object[])	
CreateInstanceFrom(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFromAnd Unwrap(String, String)	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFromAnd Unwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly file, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.
CreateInstanceFromAnd Unwrap(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly file.
CreateObjRef(Type)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (Inherited from MarshalByRefObject)
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
ExecuteAssembly(String)	Executes the assembly contained in the specified file.
ExecuteAssembly(String, String[])	Executes the assembly contained in the specified file, using the specified arguments.
ExecuteAssembly(String, String[], Byte[], AssemblyHashAlgorithm)	**Obsolete.** Executes the assembly contained in the specified file, using the specified arguments, hash value, and hash algorithm.
ExecuteAssemblyByName(Assembly Name, String[])	Executes the assembly given an AssemblyName, using the specified arguments.
ExecuteAssemblyByName(String)	Executes an assembly given its display name.
ExecuteAssemblyByName(String, String[])	Executes the assembly given its display name, using the specified arguments.
GetAssemblies()	Gets the assemblies that have been loaded into the execution context of this application domain.
GetCurrentThreadId()	**Obsolete.** Gets the current thread identifier.

GetData(String)	Gets the value stored in the current application domain for the specified name.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetLifetimeService()	**Obsolete.** Retrieves the current lifetime service object that controls the lifetime policy for this instance. (Inherited from MarshalByRefObject)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeLifetimeService()	**Obsolete.** Obtains a lifetime service object to control the lifetime policy for this instance. (Inherited from MarshalByRefObject)
IsCompatibilitySwitchSet(String)	Gets a nullable Boolean value that indicates whether any compatibility switches are set, and if so, whether the specified compatibility switch is set.
IsDefaultAppDomain()	Returns a value that indicates whether the application domain is the default application domain for the process.
IsFinalizingForUnload()	Indicates whether this application domain is unloading, and the objects it contains are being finalized by the common language runtime.
Load(AssemblyName)	Loads an Assembly given its AssemblyName.
Load(Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly.
Load(Byte[], Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly. The raw bytes representing the symbols for the Assembly are also loaded.
Load(String)	Loads an Assembly given its display name.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
MemberwiseClone(Boolean)	Creates a shallow copy of the current MarshalByRefObject object. (Inherited from MarshalByRefObject)

ReflectionOnlyGetAssemblies()	Returns the assemblies that have been loaded into the reflection-only context of the application domain.
SetCachePath(String)	**Obsolete.** Establishes the specified directory path as the location where assemblies are shadow copied.
SetData(String, Object)	Assigns the specified value to the specified application domain property.
SetDynamicBase(String)	**Obsolete.** Establishes the specified directory path as the base directory for subdirectories where dynamically generated files are stored and accessed.
SetPrincipalPolicy(PrincipalPolicy)	Specifies how principal and identity objects should be attached to a thread if the thread attempts to bind to a principal while executing in this application domain.
SetShadowCopyFiles()	**Obsolete.** Turns on shadow copying.
SetShadowCopyPath(String)	**Obsolete.** Establishes the specified directory path as the location of assemblies to be shadow copied.
SetThreadPrincipal(IPrincipal)	Sets the default principal object to be attached to threads if they attempt to bind to a principal while executing in this application domain.
ToString()	Obtains a string representation that includes the friendly name of the application domain and any context policies.
Unload(AppDomain)	**Obsolete.** Unloads the specified application domain.

Events

Expand table

AssemblyLoad	Occurs when an assembly is loaded.
AssemblyResolve	Occurs when the resolution of an assembly fails.
DomainUnload	Occurs when an AppDomain is about to be unloaded.

FirstChance Exception	Occurs when an exception is thrown in managed code, before the runtime searches the call stack for an exception handler in the application domain.
ProcessExit	Occurs when the default application domain's parent process exits.
ReflectionOnly AssemblyResolve	Occurs when the resolution of an assembly fails in the reflection-only context.
ResourceResolve	Occurs when the resolution of a resource fails because the resource is not a valid linked or embedded resource in the assembly.
TypeResolve	Occurs when the resolution of a type fails.
UnhandledException	Occurs when an exception is not caught.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- How To: Configure an Application Domain
- How To: Create an Application Domain
- How to: Load Assemblies into an Application Domain
- How to: Unload an Application Domain

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.BaseDirectory Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the base directory that the assembly resolver uses to probe for assemblies.

C#

```
public string BaseDirectory { get; }
```

Property Value

String

The base directory that the assembly resolver uses to probe for assemblies.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following code example creates a new application domain, specifying a base directory to use when searching for assemblies to load into the domain. The example then uses the BaseDirectory property to obtain the base directory path, for display to the console.

C#

```
using System;
using System.IO;
using System.Reflection;
using System.Security.Policy;

class ADSetupSnippet
{
    public static void Main()
```

```
    {
        // Create application domain setup information
        var domaininfo = new AppDomainSetup();
        domaininfo.ConfigurationFile =
Path.Combine(System.Environment.CurrentDirectory, "ADSetup.exe.config");
        domaininfo.ApplicationBase = System.Environment.CurrentDirectory;

        //Create evidence for the new appdomain from evidence of the current
application domain
        Evidence adEvidence = AppDomain.CurrentDomain.Evidence;

        // Create appdomain
        AppDomain domain = AppDomain.CreateDomain("Domain2", adEvidence,
domaininfo);

        // Display application domain information.
        Console.WriteLine("Host domain: " +
AppDomain.CurrentDomain.FriendlyName);
        Console.WriteLine("Child domain: " + domain.FriendlyName);
        Console.WriteLine();
        Console.WriteLine("Configuration file: " +
domain.SetupInformation.ConfigurationFile);
        Console.WriteLine("Application Base Directory: " +
domain.BaseDirectory);

        AppDomain.Unload(domain);
    }
}
// The example displays output like the following:
//    Host domain: adsetup.exe
//    Child domain: Domain2
//
//    Configuration file: C:\Test\ADSetup.exe.config
//    Application Base Directory: C:\Test
```

Remarks

This property corresponds to the AppDomainSetup.ApplicationBase property. It can also be retrieved using the GetData method with the string "APPBASE".

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ApplicationBase

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CurrentDomain Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current application domain for the current Thread.

C#

```
public static AppDomain CurrentDomain { get; }
```

Property Value

AppDomain

The current application domain.

Examples

The following code example creates a new application domain. The CurrentDomain property is used to obtain an AppDomain object that represents the current application domain. The FriendlyName property provides the name of the current application domain, which is then displayed at the command line.

C#

```
using System;

class ADSetupInformation
{
    static void Main()
    {
        AppDomain root = AppDomain.CurrentDomain;

        AppDomainSetup setup = new AppDomainSetup();
        setup.ApplicationBase =
            root.SetupInformation.ApplicationBase + @"MyAppSubfolder\";

        AppDomain domain = AppDomain.CreateDomain("MyDomain", null, setup);

        Console.WriteLine("Application base of {0}:\r\n\t{1}",
```

```
            root.FriendlyName, root.SetupInformation.ApplicationBase);
        Console.WriteLine("Application base of {0}:\r\n\t{1}",
            domain.FriendlyName, domain.SetupInformation.ApplicationBase);

        AppDomain.Unload(domain);
    }
}

/* This example produces output similar to the following:

Application base of MyApp.exe:
        C:\Program Files\MyApp\
Application base of MyDomain:
        C:\Program Files\MyApp\MyAppSubfolder\
 */
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.DynamicDirectory Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the directory that the assembly resolver uses to probe for dynamically created assemblies.

C#

```
public string? DynamicDirectory { get; }
```

Property Value

String

The directory that the assembly resolver uses to probe for dynamically created assemblies.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following example creates an application domain with a directory for dynamic assemblies, emits a dynamic assembly and stores it in the dynamic directory, and then loads the assembly into the new application domain and uses it.

The example creates an AppDomainSetup object and sets its ApplicationName property to "Example" and its DynamicBase property to "C:\DynamicAssemblyDir". The example then displays the DynamicBase property, to show that the hash code of the application name has been appended as a subdirectory of the path that was originally assigned.

Note

The base directory in this example is intended to be outside the probing path for the example application. Be sure to compile the example in a different location. Delete the base directory and all its subdirectories each time you run the example.

The example creates a new application domain, using the AppDomainSetup object. The example uses the DynamicDirectory property to retrieve the name of the directory, so it can create the directory. (The example could just as easily create the directory beforehand by concatenating the original path, the hash code of the application name, and the application name.)

The example has a `GenerateDynamicAssembly` method that emits an assembly named `DynamicHelloWorld.dll` and stores it in the new application domain's dynamic directory. The dynamic assembly contains one type, `HelloWorld`, that has a static method (`Shared` method in Visual Basic) named `HelloFromAD`. Calling this method displays the name of the application domain.

The `Example` class derives from MarshalByRefObject, so the example can create an instance of the class in the new application domain and call its `Test` method. The `Test` method loads the dynamic assembly by its display name and calls the static `HelloFromAD` method.

You can show that the dynamic directory is searched after the normal probing paths by writing code for an assembly named `DynamicHelloWorld.dll` and compiling it in the same directory as this example. The assembly must have a class named `HelloWorld` with a static method named `HelloFromAD`. This method does not have to have the same functionality as the one in the example; it can simply display a string to the console. The assembly must also have an AssemblyVersionAttribute attribute that sets its version to 1.0.0.0. When you run the example, the assembly you compiled in the current directory is found before the dynamic directory is searched.

C#

```
using System;
using System.Reflection;
using System.Reflection.Emit;

public class AddDynamicBaseSnippet : MarshalByRefObject
{
   static void Main()
   {
      // Prepare to create a new application domain.
      AppDomainSetup setup = new AppDomainSetup();

      // Set the application name before setting the dynamic base.
```

```
      setup.ApplicationName = "Example";

      // Set the location of the base directory where assembly resolution
      // probes for dynamic assemblies. Note that the hash code of the
      // application name is concatenated to the base directory name you
      // supply.
      setup.DynamicBase = "C:\\DynamicAssemblyDir";
      Console.WriteLine("DynamicBase is set to '{0}'.", setup.DynamicBase);

      AppDomain ad = AppDomain.CreateDomain("MyDomain", null, setup);

      // The dynamic directory name is the dynamic base concatenated with
      // the application name: <DynamicBase>\<hash code>\<ApplicationName>
      string dynamicDir = ad.DynamicDirectory;
      Console.WriteLine("Dynamic directory is '{0}'.", dynamicDir);

      // The AssemblyBuilder won't create this directory automatically.
      if (!System.IO.Directory.Exists(dynamicDir))
      {
         Console.WriteLine("Creating the dynamic directory.");
         System.IO.Directory.CreateDirectory(dynamicDir);
      }

      // Generate a dynamic assembly and store it in the dynamic
      // directory.
      GenerateDynamicAssembly(dynamicDir);

      // Create an instance of the Example class in the application domain,
      // and call its Test method to load the dynamic assembly and use it.
      AddDynamicBaseSnippet ex = (AddDynamicBaseSnippet)
ad.CreateInstanceAndUnwrap(
         typeof(AddDynamicBaseSnippet).Assembly.FullName, "Example");
      ex.Test();
   }

   public void Test()
   {
      Assembly dynAssem = Assembly.Load(
         "DynamicHelloWorld, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null");

      Type myType = dynAssem.GetType("HelloWorld");
      myType.InvokeMember("HelloFromAD", BindingFlags.Public |
         BindingFlags.Static | BindingFlags.InvokeMethod,
         Type.DefaultBinder, null, null);
   }

   private static void GenerateDynamicAssembly(string location)
   {
      // Define the dynamic assembly and the module. There is only one
      // module in this assembly. Note that the call to
DefineDynamicAssembly
      // specifies the location where the assembly will be saved. The
      // assembly version is 1.0.0.0.
      //
```

```
        AssemblyName asmName = new AssemblyName("DynamicHelloWorld");
        asmName.Version = new Version("1.0.0.0");

        AssemblyBuilder ab =
           AppDomain.CurrentDomain.DefineDynamicAssembly(
              asmName, AssemblyBuilderAccess.Save, location);

        String moduleName = asmName.Name + ".exe";
        ModuleBuilder mb = ab.DefineDynamicModule(asmName.Name, moduleName);

        // Define the "HelloWorld" type, with one static method.
        TypeBuilder tb = mb.DefineType("HelloWorld", TypeAttributes.Public);
        MethodBuilder hello = tb.DefineMethod("HelloFromAD",
           MethodAttributes.Public | MethodAttributes.Static, null, null);

        // The method displays a message that contains the name of the
application
        // domain where the method is executed.
        ILGenerator il = hello.GetILGenerator();
        il.Emit(OpCodes.Ldstr, "Hello from '{0}'!");
        il.Emit(OpCodes.Call,
typeof(AppDomain).GetProperty("CurrentDomain").GetGetMethod());
        il.Emit(OpCodes.Call,
typeof(AppDomain).GetProperty("FriendlyName").GetGetMethod());
        il.Emit(OpCodes.Call, typeof(Console).GetMethod("WriteLine",
                                 new Type[] { typeof(String), typeof(String)
}));
        il.Emit(OpCodes.Ret);

        // Complete the HelloWorld type and save the assembly. The assembly
        // is placed in the location specified by DefineDynamicAssembly.
        Type myType = tb.CreateType();
        ab.Save(moduleName);
    }
}

/* This example produces output similar to the following:

DynamicBase is set to 'C:\DynamicAssemblyDir\5e4a7545'.
Dynamic directory is 'C:\DynamicAssemblyDir\5e4a7545\Example'.
Creating the dynamic directory.
Hello from 'MyDomain'!
 */
```

Remarks

To set the dynamic directory, assign a base directory path to the AppDomainSetup.DynamicBase property of the AppDomainSetup object that will be used to create the new application domain. The base directory path you assign to the property is modified by the addition of a subdirectory whose simple name is the hash code of the string you assign to the AppDomainSetup.ApplicationName property, so the

format of the base directory is *original path\hash code*. The dynamic directory is a subdirectory of this base directory. Its simple name is the value of the AppDomainSetup.ApplicationName property, so its format is *original path\hash code\application name*.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.FriendlyName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the friendly name of this application domain.

C#

```
public string FriendlyName { get; }
```

Property Value

String

The friendly name of this application domain.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following code example uses the FriendlyName property to get the friendly name of the current application domain. For the default application domain, the friendly name is the name of the application's executable file. The code example also displays additional information about the application domain.

C#

```
using System;

class ADSetupInformation
{
    static void Main()
    {
        AppDomain root = AppDomain.CurrentDomain;
```

```
        AppDomainSetup setup = new AppDomainSetup();
        setup.ApplicationBase =
            root.SetupInformation.ApplicationBase + @"MyAppSubfolder\";

        AppDomain domain = AppDomain.CreateDomain("MyDomain", null, setup);

        Console.WriteLine("Application base of {0}:\r\n\t{1}",
            root.FriendlyName, root.SetupInformation.ApplicationBase);
        Console.WriteLine("Application base of {0}:\r\n\t{1}",
            domain.FriendlyName, domain.SetupInformation.ApplicationBase);

        AppDomain.Unload(domain);
    }
}

/* This example produces output similar to the following:

Application base of MyApp.exe:
        C:\Program Files\MyApp\
Application base of MyDomain:
        C:\Program Files\MyApp\MyAppSubfolder\
 */
```

Remarks

The friendly name of the default application domain is the file name of the process executable. For example, if the executable used to start the process is `"c:\MyAppDirectory\MyAssembly.exe"`, the friendly name of the default application domain is `"MyAssembly.exe"`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

AppDomain.Id Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an integer that uniquely identifies the application domain within the process.

C#

```
public int Id { get; }
```

Property Value

Int32

An integer that identifies the application domain.

Examples

The following code example creates a second application domain and displays information about the default domain and the new domain.

C#

```
using System;
using System.Reflection;

public class AppDomainIdSnippet
{
    // The following attribute indicates to the loader that assemblies
    // in the global assembly cache should be shared across multiple
    // application domains.
    [LoaderOptimizationAttribute( LoaderOptimization.MultiDomainHost)]
    public static void Main()
    {
        // Show information for the default application domain.
        ShowDomainInfo();

        // Create a new application domain and display its information.
        AppDomain newDomain = AppDomain.CreateDomain("MyMultiDomain");
        newDomain.DoCallBack(new CrossAppDomainDelegate(ShowDomainInfo));
```

```
    }

    // This method has the same signature as the CrossAppDomainDelegate,
    // so that it can be executed easily in the new application domain.
    //
    public static void ShowDomainInfo()
    {
        AppDomain ad = AppDomain.CurrentDomain;
        Console.WriteLine();
        Console.WriteLine("FriendlyName: {0}", ad.FriendlyName);
        Console.WriteLine("Id: {0}", ad.Id);
        Console.WriteLine("IsDefaultAppDomain: {0}",
ad.IsDefaultAppDomain());
    }
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.IsFullyTrusted Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether assemblies that are loaded into the current application domain execute with full trust.

```
C#
public bool IsFullyTrusted { get; }
```

Property Value

Boolean

`true` if assemblies that are loaded into the current application domain execute with full trust; otherwise, `false`.

Examples

The following example demonstrates the IsFullyTrusted property and the Assembly.IsFullyTrusted property with fully trusted and partially trusted application domains. The fully trusted application domain is the default application domain for the application. The partially trusted application domain is created by using the AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[]) method overload.

The example uses a `Worker` class that derives from MarshalByRefObject, so it can be marshaled across application domain boundaries. The example creates a `Worker` object in the default application domain. It then calls the `TestIsFullyTrusted` method to display the property value for the application domain and for two assemblies that are loaded into the application domain: mscorlib, which is part of the .NET Framework, and the example assembly. The application domain is fully trusted, so both assemblies are fully trusted.

The example creates another `Worker` object in a sandboxed application domain and again calls the `TestIsFullyTrusted` method. Mscorlib is always trusted, even in a partially trusted application domain, but the example assembly is partially trusted.

C#

```
using System;

namespace SimpleSandboxing
{
    public class Worker : MarshalByRefObject
    {
        static void Main()
        {
            Worker w = new Worker();
            w.TestIsFullyTrusted();

            AppDomain adSandbox = GetInternetSandbox();
            w = (Worker) adSandbox.CreateInstanceAndUnwrap(
                                typeof(Worker).Assembly.FullName,
                                typeof(Worker).FullName);
            w.TestIsFullyTrusted();
        }

        public void TestIsFullyTrusted()
        {
            AppDomain ad = AppDomain.CurrentDomain;
            Console.WriteLine("\r\nApplication domain '{0}': IsFullyTrusted
= {1}",
                                        ad.FriendlyName, ad.IsFullyTrusted);

            Console.WriteLine("   IsFullyTrusted = {0} for the current
assembly",
                              typeof(Worker).Assembly.IsFullyTrusted);

            Console.WriteLine("   IsFullyTrusted = {0} for mscorlib",

typeof(int).Assembly.IsFullyTrusted);
        }

        // ------------ Helper method -------------------------------------
-
        static AppDomain GetInternetSandbox()
        {
            // Create the permission set to grant to all assemblies.
            System.Security.Policy.Evidence hostEvidence = new
System.Security.Policy.Evidence();
            hostEvidence.AddHostEvidence(new System.Security.Policy.Zone(

System.Security.SecurityZone.Internet));
            System.Security.PermissionSet pset =

System.Security.SecurityManager.GetStandardSandbox(hostEvidence);
```

```
            // Identify the folder to use for the sandbox.
            AppDomainSetup ads = new AppDomainSetup();
            ads.ApplicationBase = System.IO.Directory.GetCurrentDirectory();

            // Create the sandboxed application domain.
            return AppDomain.CreateDomain("Sandbox", hostEvidence, ads,
pset, null);
        }
    }
}

/* This example produces output similar to the following:

Application domain 'Example.exe': IsFullyTrusted = True
   IsFullyTrusted = True for the current assembly
   IsFullyTrusted = True for mscorlib

Application domain 'Sandbox': IsFullyTrusted = False
   IsFullyTrusted = False for the current assembly
   IsFullyTrusted = True for mscorlib
 */
```

Remarks

This method always returns `true` for the default application domain of an application that runs on the desktop. It returns `false` for a sandboxed application domain that was created by using the AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[]) method overload, unless the permissions that are granted to the application domain are equivalent to full trust.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.IsHomogenous Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current application domain has a set of permissions that is granted to all assemblies that are loaded into the application domain.

```
C#

public bool IsHomogenous { get; }
```

Property Value

Boolean

`true` if the current application domain has a homogenous set of permissions; otherwise, `false`.

Remarks

This property returns `true` for sandboxed application domains that were created by using the AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[]) method overload. Sandboxed application domains have a homogenous set of permissions; that is, the same set of permissions is granted to all partially trusted assemblies that are loaded into the application domain. A sandboxed application domain optionally has a list of strong-named assemblies that are exempt from this permission set, and instead run with full trust.

Fully trusted code can use the PermissionSet property to determine the homogenous grant set of a sandboxed application domain.

This property also returns `true` for the default application domain of a desktop application, because that application domain grants full trust to all assemblies.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- PermissionSet

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.MonitoringIsEnabled Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets a value that indicates whether CPU and memory monitoring of application domains is enabled for the current process. Once monitoring is enabled for a process, it cannot be disabled.

```
C#
public static bool MonitoringIsEnabled { get; set; }
```

Property Value

Boolean

`true` if monitoring is enabled; otherwise `false`.

Exceptions

ArgumentException

The current process attempted to assign the value `false` to this property.

Remarks

This `static` property (`Shared` property in Visual Basic) controls CPU and memory monitoring of all the application domains in the process.

If you attempt to set this property to `false`, a ArgumentException exception is thrown, even if the current value of the property is `false`.

Once monitoring is enabled, you can use the MonitoringSurvivedMemorySize, MonitoringSurvivedProcessMemorySize, MonitoringTotalAllocatedMemorySize, and

MonitoringTotalProcessorTime instance properties to monitor CPU and memory use of individual application domains.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Application Domain Resource Monitoring
- <appdomainResourceMonitoring> Element

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.MonitoringSurvivedMemorySize Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that survived the last collection and that are known to be referenced by the current application domain.

```
C#

public long MonitoringSurvivedMemorySize { get; }
```

Property Value

Int64

The number of surviving bytes.

Exceptions

InvalidOperationException

The `static` (`Shared` in Visual Basic) MonitoringIsEnabled property is set to `false`.

Remarks

The statistics are updated with each garbage collection. However, they are guaranteed to be accurate only after a full, blocking collection; that is, a collection that includes all generations and that stops the application while collection occurs. For example, the GC.Collect() method overload performs a full, blocking collection. (Concurrent collection occurs in the background and does not block the application.)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Application Domain Resource Monitoring

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.MonitoringSurvivedProcessMemorySize Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total bytes that survived from the last collection for all application domains in the process.

C#

```
public static long MonitoringSurvivedProcessMemorySize { get; }
```

Property Value

Int64

The total number of surviving bytes for the process.

Exceptions

InvalidOperationException

The `static` (`Shared` in Visual Basic) MonitoringIsEnabled property is set to `false`.

Remarks

After a full, blocking collection, this number represents the number of bytes currently held live on managed heaps. It should be close to the number reported by the GetTotalMemory method. After an ephemeral collection, this number represents the number of bytes currently held live in ephemeral generations.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Application Domain Resource Monitoring

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.MonitoringTotalAllocatedMemorySize Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total size, in bytes, of all memory allocations that have been made by the application domain since it was created, without subtracting memory that has been collected.

C#

```
public long MonitoringTotalAllocatedMemorySize { get; }
```

Property Value

Int64

The total size of all memory allocations.

Exceptions

InvalidOperationException

The `static` (`Shared` in Visual Basic) MonitoringIsEnabled property is set to `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Application Domain Resource Monitoring

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.MonitoringTotalProcessorTime Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total processor time that has been used by all threads while executing in the current application domain, since the process started.

```
C#

public TimeSpan MonitoringTotalProcessorTime { get; }
```

Property Value

TimeSpan

Total processor time for the current application domain.

Exceptions

InvalidOperationException

The `static` (`Shared` in Visual Basic) MonitoringIsEnabled property is set to `false`.

Remarks

The total time that is reported for an application domain includes the time each thread in the process spent executing in that application domain.

A thread that calls into unmanaged code is still associated with an application domain, and the processor time spent executing the unmanaged code is reported for the application domain where the call was made.

When a thread is blocked or sleeping, it does not consume processor time.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Application Domain Resource Monitoring

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.PermissionSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Code Access Security is not supported or honored by the runtime.

Gets the permission set of a sandboxed application domain.

C#

```
[System.Obsolete("Code Access Security is not supported or honored by the runtime.", DiagnosticId="SYSLIB0003", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public System.Security.PermissionSet PermissionSet { get; }
```

Property Value

PermissionSet

The permission set of the sandboxed application domain.

Attributes ObsoleteAttribute

Remarks

> ⊗ **Caution**
>
> Code Access Security (CAS) has been deprecated across all versions of .NET Framework and .NET. Recent versions of .NET do not honor CAS annotations and produce errors if CAS-related APIs are used. Developers should seek alternative means of accomplishing security tasks.

Sandboxed application domains that were created by using the AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[]) method overload have a homogenous set of permissions; that is, the same set of permissions is granted to all partially trusted assemblies that are loaded into the application domain. A sandboxed application domain optionally has a list of strong-named assemblies that are exempt from this permission set, and instead run with full trust.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 3.0, Core 3.1 *(5, 6, 7, 8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

See also

- IsHomogenous

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.RelativeSearchPath Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the path under the base directory where the assembly resolver should probe for private assemblies.

C#

```
public string? RelativeSearchPath { get; }
```

Property Value

String

The path under the base directory where the assembly resolver should probe for private assemblies.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Remarks

Private assemblies are deployed in the same directory structure as the application. If the path specified by the RelativeSearchPath property is not under AppDomainSetup.ApplicationBase, it is ignored.

This property returns the value set using AppDomainSetup.PrivateBinPath.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- PrivateBinPath

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetupInformation Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the application domain configuration information for this instance.

```
C#
public AppDomainSetup SetupInformation { get; }
```

Property Value

AppDomainSetup

The application domain initialization information.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

AppDomain.ShadowCopyFiles Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an indication whether the application domain is configured to shadow copy files.

C#

```
public bool ShadowCopyFiles { get; }
```

Property Value

Boolean

`true` if the application domain is configured to shadow copy files; otherwise, `false`.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Remarks

For more information, see AppDomainSetup.ShadowCopyFiles and Shadow Copying Assemblies.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ShadowCopyFiles
- Shadow Copying Assemblies

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.AppendPrivatePath(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.AppendPrivatePath has been deprecated and is not supported.

Appends the specified directory name to the private path list.

C#

```
[System.Obsolete("AppDomain.AppendPrivatePath has been deprecated and is not supported.")]
public void AppendPrivatePath (string? path);
```

Parameters

path String

The name of the directory to be appended to the private path.

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Remarks

The use of this property is not recommended, because it might change the probing path for assemblies after they have already been loaded. Use the

AppDomainSetup.PrivateBinPath property instead.

The private path, or relative search path, is the path relative to the base directory where the assembly resolver probes for private assemblies.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ApplyPolicy(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the assembly display name after policy has been applied.

C#

```
public string ApplyPolicy (string assemblyName);
```

Parameters

`assemblyName` String

The assembly display name, in the form provided by the FullName property.

Returns

String

A string containing the assembly display name after policy has been applied.

Remarks

The ApplyPolicy method takes an assembly display name and returns the post-policy display name. This is useful if you need to load an assembly using policy, because the reflection-only context does not apply policy.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.ClearPrivatePath Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.ClearPrivatePath has been deprecated and is not supported.

Resets the path that specifies the location of private assemblies to the empty string ("").

C#

```
[System.Obsolete("AppDomain.ClearPrivatePath has been deprecated and is not supported.")]
public void ClearPrivatePath ();
```

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following code example demonstrates how to use the ClearPrivatePath method to remove all entries from the list of private paths to search when assemblies are loaded.

This method is now obsolete, and should not be used for new development.

C#

```
using System;
using System.Reflection;
using System.Security.Policy;
```

```
class ADAppendPrivatePath
{
    public static void Main()
    {
        //Create evidence for new appdomain.
        Evidence adevidence = AppDomain.CurrentDomain.Evidence;

        //Create the new application domain.
        AppDomain domain = AppDomain.CreateDomain("MyDomain", adevidence);

        //Display the current relative search path.
        Console.WriteLine("Relative search path is: " +
domain.RelativeSearchPath);

        //Append the relative path.
        String Newpath = "www.code.microsoft.com";
        domain.AppendPrivatePath(Newpath);

        //Display the new relative search path.
        Console.WriteLine("Relative search path is: " +
domain.RelativeSearchPath);

        //Clear the private search path.
        domain.ClearPrivatePath();

        //Display the new relative search path.
        Console.WriteLine("Relative search path is now: " +
domain.RelativeSearchPath);

        AppDomain.Unload(domain);
    }
}
```

Remarks

The private path is a path relative to the base directory that the common language runtime searches to locate private assemblies.

For more information, see AppDomainSetup.PrivateBinPath.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- Empty

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ClearShadowCopyPath Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.ClearShadowCopyPath has been deprecated and is not supported.

Resets the list of directories containing shadow copied assemblies to the empty string ("").

C#

```
[System.Obsolete("AppDomain.ClearShadowCopyPath has been deprecated and is not supported.")]
public void ClearShadowCopyPath ();
```

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Remarks

The shadow copy path is a list of directories where shadow copied assemblies are stored.

For more information, see AppDomainSetup.ShadowCopyDirectories and Shadow Copying Assemblies.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- Empty
- Shadow Copying Assemblies

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CreateDomain Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new application domain.

Overloads

Expand table

CreateDomain(String)	**Obsolete.** Creates a new application domain with the specified name.

CreateDomain(String)

Caution

Creating and unloading AppDomains is not supported and throws an exception.

Creates a new application domain with the specified name.

C#

```
[System.Obsolete("Creating and unloading AppDomains is not supported and throws an exception.", DiagnosticId="SYSLIB0024", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public static AppDomain CreateDomain (string friendlyName);
```

Parameters

`friendlyName` String

The friendly name of the domain.

Returns

AppDomain

The newly created application domain.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`friendlyName` is `null`.

PlatformNotSupportedException

.NET Core and .NET 5+ only: In all cases.

Examples

The following sample demonstrates, in general, how to create a domain using one of the CreateDomain overloads.

C#

```
// Set up the AppDomainSetup
AppDomainSetup setup = new AppDomainSetup();
setup.ApplicationBase = "(some directory)";
setup.ConfigurationFile = "(some file)";

// Set up the Evidence
Evidence baseEvidence = AppDomain.CurrentDomain.Evidence;
Evidence evidence = new Evidence(baseEvidence);
evidence.AddAssembly("(some assembly)");
evidence.AddHost("(some host)");

// Create the AppDomain
AppDomain newDomain = AppDomain.CreateDomain("newDomain", evidence,
setup);
```

Remarks

The `friendlyName` parameter is intended to identify the domain in a manner that is meaningful to humans. This string should be suitable for display in user interfaces.

This method overload uses the AppDomainSetup information from the default application domain.

See also

- FriendlyName
- Unavailable technologies on .NET Core/5+: Application domains

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CreateInstance Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new instance of a specified type defined in a specified assembly.

Overloads

Expand table

CreateInstance(String, String)	Creates a new instance of the specified type defined in the specified assembly.
CreateInstance(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly. A parameter specifies an array of activation attributes.
CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly. Parameters specify a binder, binding flags, constructor arguments, culture-specific information used to interpret arguments, and optional activation attributes.

CreateInstance(String, String)

Creates a new instance of the specified type defined in the specified assembly.

C#

```
public System.Runtime.Remoting.ObjectHandle? CreateInstance (string
assemblyName, string typeName);
```

Parameters

assemblyName String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

Returns

ObjectHandle

An object that is a wrapper for the new instance specified by `typeName`. The return value needs to be unwrapped to access the real object.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

ArgumentNullException

`assemblyName` or `typeName` is `null`.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

NullReferenceException

This instance is `null`.

Remarks

This method calls the parameterless constructor for `typeName`.

See AssemblyName for the format of `assemblyName`.

An attempt to call CreateInstance on a target application domain that is not the current application domain will result in a successful load of the assembly in the target application domain. Since an Assembly is not MarshalByRefObject, when this method attempts to return the Assembly for the loaded assembly to the current application domain, the common language runtime will try to load the assembly into the current application domain and the load might fail. The assembly that is loaded into the current application domain might be different from the assembly that was loaded first if the path settings for the two application domains are different.

See also

- CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstance(String, String, Object[])

Creates a new instance of the specified type defined in the specified assembly. A parameter specifies an array of activation attributes.

C#

```
public System.Runtime.Remoting.ObjectHandle? CreateInstance (string
```

```
assemblyName, string typeName, object?[]? activationAttributes);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects.Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

An object that is a wrapper for the new instance specified by `typeName`. The return value needs to be unwrapped to access the real object.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

ArgumentNullException

`assemblyName` or `typeName` is `null`.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MissingMethodException

No matching public constructor was found.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

TypeLoadException

`typename` was not found in `assemblyName`.

NullReferenceException

This instance is `null`.

Remarks

This method calls the parameterless constructor for `typeName`.

See AssemblyName for the format of `assemblyName`.

An attempt to call CreateInstance on a target application domain that is not the current application domain will result in a successful load of the assembly in the target application domain. Since an Assembly is not MarshalByRefObject, when this method attempts to return the Assembly for the loaded assembly to the current application domain, the common language runtime will try to load the assembly into the current application domain and the load might fail. The assembly that is loaded into the current application domain might be different from the assembly that was loaded first if the path settings for the two application domains are different.

See also

- CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates a new instance of the specified type defined in the specified assembly. Parameters specify a binder, binding flags, constructor arguments, culture-specific information used to interpret arguments, and optional activation attributes.

C#

```
public System.Runtime.Remoting.ObjectHandle? CreateInstance (string
assemblyName, string typeName, bool ignoreCase,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, object?[]? args, System.Globalization.CultureInfo? culture,
object?[]? activationAttributes);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`ignoreCase` Boolean

A Boolean value specifying whether to perform a case-sensitive search or not.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that enables the binding, coercion of argument types, invocation of members, and retrieval of MemberInfo objects using reflection. If `binder` is null, the default binder is used.

`args` Object[]

The arguments to pass to the constructor. This array of arguments must match in number, order, and type the parameters of the constructor to invoke. If the parameterless constructor is preferred, `args` must be an empty array or null.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `typeName` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

An object that is a wrapper for the new instance specified by `typeName`. The return value needs to be unwrapped to access the real object.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

ArgumentNullException

`assemblyName` or `typeName` is `null`.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

`assemblyName` was compiled with a later version of the common language runtime than the version that is currently loaded.

FileLoadException

An assembly or module was loaded twice with two different evidences.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

MissingMethodException

No matching constructor was found.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

TypeLoadException

`typename` was not found in `assemblyName`.

NullReferenceException

This instance is `null`.

Remarks

See AssemblyName for the format of `assemblyName`.

An attempt to call CreateInstance on a target application domain that is not the current application domain will result in a successful load of the assembly in the target application domain. Since an Assembly is not MarshalByRefObject, when this method attempts to return the Assembly for the loaded assembly to the current application domain, the common language runtime will try to load the assembly into the current application domain and the load might fail. The assembly that is loaded into the current application domain might be different from the assembly that was loaded first if the path settings for the two application domains are different.

See also

- CreateInstance(Type, BindingFlags, Binder, Object[], CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CreateInstanceAndUnwrap Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new instance of a specified type.

Overloads

Expand table

CreateInstanceAndUnwrap(String, String)	Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, and the name of the type.
CreateInstanceAndUnwrap(String, String, Object[])	Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, the name of the type, and an array of activation attributes.
CreateInstanceAndUnwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.

CreateInstanceAndUnwrap(String, String)

Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, and the name of the type.

C#

```
public object? CreateInstanceAndUnwrap (string assemblyName, string
```

```
typeName);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

Returns

Object

An instance of the object specified by `typeName`.

Exceptions

ArgumentNullException

`assemblyName` or `typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following code example shows the simplest way to execute code in another application domain. The example defines a class named `Worker` that inherits from MarshalByRefObject. The `Worker` class defines a method that displays the name of the application domain in which it is executing. The example creates instances of `Worker` in the default application domain and in a new application domain.

> **Note**
>
> The assembly that contains `Worker` must be loaded into both application domains, but it can load other assemblies that exist only in the new application domain.

C#

```
using System;
using System.Reflection;

public class CreateInstanceWorker : MarshalByRefObject
{
    public void PrintDomain()
    {
        Console.WriteLine("Object is executing in AppDomain \"{0}\"",
            AppDomain.CurrentDomain.FriendlyName);
    }
}

class CreateInstanceAndUnwrapSourceSnippet
{
    public static void Main()
    {
        // Create an ordinary instance in the current AppDomain
        CreateInstanceWorker localWorker = new CreateInstanceWorker();
        localWorker.PrintDomain();

        // Create a new application domain, create an instance
        // of Worker in the application domain, and execute code
        // there.
        AppDomain ad = AppDomain.CreateDomain("New domain");
```

```
        CreateInstanceWorker remoteWorker = (CreateInstanceWorker)
ad.CreateInstanceAndUnwrap(
            typeof(CreateInstanceWorker).Assembly.FullName,
            "Worker");
        remoteWorker.PrintDomain();
    }
}

/* This code produces output similar to the following:

Object is executing in AppDomain "source.exe"
Object is executing in AppDomain "New domain"
 */
```

Remarks

This is a convenience method that combines CreateInstance and ObjectHandle.Unwrap. This method calls the parameterless constructor for `typeName`.

See AssemblyName for the format of `assemblyName`. See the Type.FullName property for the format of `typeName`.

> **Note**
>
> If you make an early-bound call to a method `M` of an object of type `T1` that was returned by **CreateInstanceAndUnwrap**, and that method makes an early-bound call to a method of an object of type `T2` in an assembly `C` other than the current assembly or the assembly containing `T1`, assembly `C` is loaded into the current application domain. This loading occurs even if the early-bound call to `T1.M()` was made in the body of a **DynamicMethod**, or in other dynamically generated code. If the current domain is the default domain, assembly `C` cannot be unloaded until the process ends. If the current domain later attempts to load assembly `C`, the load might fail.

See also

- CreateInstance(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceAndUnwrap(String, String, Object[])

Creates a new instance of the specified type. Parameters specify the assembly where the type is defined, the name of the type, and an array of activation attributes.

C#

```
public object? CreateInstanceAndUnwrap (string assemblyName, string
typeName, object?[]? activationAttributes);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects.Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

An instance of the object specified by `typeName`.

Exceptions

ArgumentNullException

`assemblyName` or `typeName` is `null`.

MissingMethodException

No matching public constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

```
C#
```

```
using System;
using System.IO;
using System.Threading;
using System.Reflection;
using System.Reflection.Emit;
using System.Runtime.Remoting;

class ADDyno

{

   public static Type CreateADynamicAssembly(ref AppDomain myNewDomain,
                        string executableNameNoExe)
   {

    string executableName = executableNameNoExe + ".exe";

    AssemblyName myAsmName = new AssemblyName();
    myAsmName.Name = executableNameNoExe;
    myAsmName.CodeBase = Environment.CurrentDirectory;

    AssemblyBuilder myAsmBuilder =
myNewDomain.DefineDynamicAssembly(myAsmName,
                        AssemblyBuilderAccess.RunAndSave);
    Console.WriteLine("-- Dynamic Assembly instantiated.");

    ModuleBuilder myModBuilder =
myAsmBuilder.DefineDynamicModule(executableNameNoExe,
                                      executableName);

    TypeBuilder myTypeBuilder =
myModBuilder.DefineType(executableNameNoExe,
                        TypeAttributes.Public,
                        typeof(MarshalByRefObject));

    MethodBuilder myFCMethod =
myTypeBuilder.DefineMethod("CountLocalFiles",
                        MethodAttributes.Public |
                        MethodAttributes.Static,
                        null,
                        new Type[] {  });

    MethodInfo currentDirGetMI =
typeof(Environment).GetProperty("CurrentDirectory").GetGetMethod();
    MethodInfo writeLine0objMI = typeof(Console).GetMethod("WriteLine",
                    new Type[] { typeof(string) });
    MethodInfo writeLine2objMI = typeof(Console).GetMethod("WriteLine",
                    new Type[] { typeof(string), typeof(object),
typeof(object) });
    MethodInfo getFilesMI = typeof(Directory).GetMethod("GetFiles",
                new Type[] { typeof(string) });

    myFCMethod.InitLocals = true;
```

```
    ILGenerator myFCIL = myFCMethod.GetILGenerator();

    Console.WriteLine("-- Generating MSIL method body...");
    LocalBuilder v0 = myFCIL.DeclareLocal(typeof(string));
    LocalBuilder v1 = myFCIL.DeclareLocal(typeof(int));
    LocalBuilder v2 = myFCIL.DeclareLocal(typeof(string));
    LocalBuilder v3 = myFCIL.DeclareLocal(typeof(string[]));

    Label evalForEachLabel = myFCIL.DefineLabel();
    Label topOfForEachLabel = myFCIL.DefineLabel();

    // Build the method body.

    myFCIL.EmitCall(OpCodes.Call, currentDirGetMI, null);
    myFCIL.Emit(OpCodes.Stloc_S, v0);
    myFCIL.Emit(OpCodes.Ldc_I4_0);
    myFCIL.Emit(OpCodes.Stloc_S, v1);
    myFCIL.Emit(OpCodes.Ldstr, "---");
    myFCIL.EmitCall(OpCodes.Call, writeLine0objMI, null);
    myFCIL.Emit(OpCodes.Ldloc_S, v0);
    myFCIL.EmitCall(OpCodes.Call, getFilesMI, null);
    myFCIL.Emit(OpCodes.Stloc_S, v3);

    myFCIL.Emit(OpCodes.Br_S, evalForEachLabel);

    // foreach loop starts here.
    myFCIL.MarkLabel(topOfForEachLabel);

        // Load array of strings and index, store value at index for
output.
    myFCIL.Emit(OpCodes.Ldloc_S, v3);
    myFCIL.Emit(OpCodes.Ldloc_S, v1);
    myFCIL.Emit(OpCodes.Ldelem_Ref);
    myFCIL.Emit(OpCodes.Stloc_S, v2);

    myFCIL.Emit(OpCodes.Ldloc_S, v2);
    myFCIL.EmitCall(OpCodes.Call, writeLine0objMI, null);

    // Increment counter by one.
    myFCIL.Emit(OpCodes.Ldloc_S, v1);
    myFCIL.Emit(OpCodes.Ldc_I4_1);
    myFCIL.Emit(OpCodes.Add);
    myFCIL.Emit(OpCodes.Stloc_S, v1);

    // Determine if end of file list array has been reached.
    myFCIL.MarkLabel(evalForEachLabel);
    myFCIL.Emit(OpCodes.Ldloc_S, v1);
    myFCIL.Emit(OpCodes.Ldloc_S, v3);
    myFCIL.Emit(OpCodes.Ldlen);
    myFCIL.Emit(OpCodes.Conv_I4);
    myFCIL.Emit(OpCodes.Blt_S, topOfForEachLabel);
    //foreach loop end here.

    myFCIL.Emit(OpCodes.Ldstr, "---");
```

```
    myFCIL.EmitCall(OpCodes.Call, writeLine0objMI, null);
    myFCIL.Emit(OpCodes.Ldstr, "There are {0} files in {1}.");
    myFCIL.Emit(OpCodes.Ldloc_S, v1);
    myFCIL.Emit(OpCodes.Box, typeof(int));
    myFCIL.Emit(OpCodes.Ldloc_S, v0);
    myFCIL.EmitCall(OpCodes.Call, writeLine2objMI, null);

    myFCIL.Emit(OpCodes.Ret);

    Type myType = myTypeBuilder.CreateType();

    myAsmBuilder.SetEntryPoint(myFCMethod);
    myAsmBuilder.Save(executableName);
    Console.WriteLine("-- Method generated, type completed, and assembly
saved to disk.");

    return myType;
   }

   public static void Main()
   {

    string domainDir, executableName = null;

    Console.Write("Enter a name for the file counting assembly: ");
    string executableNameNoExe = Console.ReadLine();
    executableName = executableNameNoExe + ".exe";
    Console.WriteLine("---");

    domainDir = Environment.CurrentDirectory;

    AppDomain curDomain = Thread.GetDomain();

    // Create a new AppDomain, with the current directory as the base.

    Console.WriteLine("Current Directory: {0}",
Environment.CurrentDirectory);
    AppDomainSetup mySetupInfo = new AppDomainSetup();
    mySetupInfo.ApplicationBase = domainDir;
    mySetupInfo.ApplicationName = executableNameNoExe;
    mySetupInfo.LoaderOptimization = LoaderOptimization.SingleDomain;

    AppDomain myDomain = AppDomain.CreateDomain(executableNameNoExe,
                    null, mySetupInfo);

    Console.WriteLine("Creating a new AppDomain '{0}'...",
                    executableNameNoExe);

    Console.WriteLine("-- Base Directory = '{0}'",
myDomain.BaseDirectory);
    Console.WriteLine("-- Shadow Copy? = '{0}'",
myDomain.ShadowCopyFiles);

    Console.WriteLine("---");
    Type myFCType = CreateADynamicAssembly(ref curDomain,
```

```
                    executableNameNoExe);

    Console.WriteLine("Loading '{0}' from '{1}'...", executableName,
              myDomain.BaseDirectory.ToString());

    BindingFlags bFlags = (BindingFlags.Public |
BindingFlags.CreateInstance |
                  BindingFlags.Instance);

    Object myObjInstance =
myDomain.CreateInstanceAndUnwrap(executableNameNoExe,
                executableNameNoExe, false, bFlags,
                null, null, null, null, null);

    Console.WriteLine("Executing method 'CountLocalFiles' in {0}...",
               myObjInstance.ToString());

    myFCType.InvokeMember("CountLocalFiles", BindingFlags.InvokeMethod,
null,
                myObjInstance, new object[] { });
   }
}
```

Remarks

This is a convenience method that combines CreateInstance and ObjectHandle.Unwrap. This method calls the parameterless constructor for `typeName`.

See AssemblyName for the format of `assemblyName`. See the Type.FullName property for the format of `typeName`.

> **Note**
>
> If you make an early-bound call to a method `M` of an object of type `T1` that was returned by **CreateInstanceAndUnwrap**, and that method makes an early-bound call to a method of an object of type `T2` in an assembly `C` other than the current assembly or the assembly containing `T1`, assembly `C` is loaded into the current application domain. This loading occurs even if the early-bound call to `T1.M()` was made in the body of a **DynamicMethod**, or in other dynamically generated code. If the current domain is the default domain, assembly `C` cannot be unloaded until the process ends. If the current domain later attempts to load assembly `C`, the load might fail.

See also

- CreateInstance(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceAndUnwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates a new instance of the specified type defined in the specified assembly, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.

C#

```
public object? CreateInstanceAndUnwrap (string assemblyName, string
typeName, bool ignoreCase, System.Reflection.BindingFlags bindingAttr,
System.Reflection.Binder? binder, object?[]? args,
System.Globalization.CultureInfo? culture, object?[]?
activationAttributes);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`ignoreCase` Boolean

A Boolean value specifying whether to perform a case-sensitive search or not.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that enables the binding, coercion of argument types, invocation of members, and retrieval of MemberInfo objects using reflection. If `binder` is null, the default binder is used.

`args` Object[]

The arguments to pass to the constructor. This array of arguments must match in number, order, and type the parameters of the constructor to invoke. If the parameterless constructor is preferred, `args` must be an empty array or null.

`culture` CultureInfo

A culture-specific object used to govern the coercion of types. If `culture` is `null`, the `CultureInfo` for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object. that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

An instance of the object specified by `typeName`.

Exceptions

ArgumentNullException

`assemblyName` or `typeName` is `null`.

MissingMethodException

No matching constructor was found.

TypeLoadException

`typename` was not found in `assemblyName`.

FileNotFoundException

`assemblyName` was not found.

MethodAccessException

The caller does not have permission to call this constructor.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

`assemblyName` was compiled with a later version of the common language runtime than the version that is currently loaded.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following sample demonstrates the use of the `ignoreCase` parameter.

C#

```
using System;
using System.Reflection;

class IgnoreCaseSnippet {

   static void Main() {
      InstantiateINT32(false);     // Failed!
      InstantiateINT32(true);      // OK!
```

```
    }

    static void InstantiateINT32(bool ignoreCase) {
        try {
            AppDomain currentDomain = AppDomain.CurrentDomain;
            object instance = currentDomain.CreateInstanceAndUnwrap(
                "mscorlib, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089",
                "SYSTEM.INT32",
                ignoreCase,
                BindingFlags.Default,
                null,
                null,
                null,
                null,
                null
            );
            Console.WriteLine(instance.GetType());
        } catch (TypeLoadException e) {
            Console.WriteLine(e.Message);
        }
    }
}
```

Remarks

This is a convenience method that combines CreateInstance and ObjectHandle.Unwrap.

See AssemblyName for the format of `assemblyName`. See the Type.FullName property for the format of `typeName`.

> **Note**
>
> If you make an early-bound call to a method `M` of an object of type `T1` that was returned by **CreateInstanceAndUnwrap**, and that method makes an early-bound call to a method of an object of type `T2` in an assembly `C` other than the current assembly or the assembly containing `T1`, assembly `C` is loaded into the current application domain. This loading occurs even if the early-bound call to `T1.M()` was made in the body of a **DynamicMethod**, or in other dynamically generated code. If the current domain is the default domain, assembly `C` cannot be unloaded until the process ends. If the current domain later attempts to load assembly `C`, the load might fail.

See also

- CreateInstance(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CreateInstanceFrom Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new instance of a specified type defined in the specified assembly file.

Overloads

Expand table

CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFrom(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFrom(String, String)	Creates a new instance of the specified type defined in the specified assembly file.

CreateInstanceFrom(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates a new instance of the specified type defined in the specified assembly file.

C#

```
public System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom (string
assemblyFile, string typeName, bool ignoreCase,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
```

```
binder, object?[]? args, System.Globalization.CultureInfo? culture,
object?[]? activationAttributes);
```

Parameters

`assemblyFile` String

The name, including the path, of a file that contains an assembly that defines the requested type. The assembly is loaded using the LoadFrom(String) method.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`ignoreCase` Boolean

A Boolean value specifying whether to perform a case-sensitive search or not.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that enables the binding, coercion of argument types, invocation of members, and retrieval of MemberInfo objects through reflection. If `binder` is null, the default binder is used.

`args` Object[]

The arguments to pass to the constructor. This array of arguments must match in number, order, and type the parameters of the constructor to invoke. If the parameterless constructor is preferred, `args` must be an empty array or null.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `typeName` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required

to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

An object that is a wrapper for the new instance, or `null` if `typeName` is not found. The return value needs to be unwrapped to access the real object.

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

-or-

`typeName` is `null`.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileNotFoundException

`assemblyFile` was not found.

TypeLoadException

`typeName` was not found in `assemblyFile`.

MissingMethodException

No matching public constructor was found.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

`assemblyFile` was compiled with a later version of the common language runtime than the version that is currently loaded.

FileLoadException

An assembly or module was loaded twice with two different evidences.

NullReferenceException

This instance is `null`.

Remarks

For more information, see the Activator.CreateInstanceFrom method.

When the CreateInstanceFrom method is used to create an instance in a target application domain, other than the application domain from which the call is made, the assembly is loaded in the target application domain. However, if the instance is unwrapped in the calling application domain, using the unwrapped instance in certain ways can cause the assembly to be loaded into the calling application domain. For example, after the instance is unwrapped, its type information might be requested, in order to call its methods late-bound. When the assembly is loaded into the calling application domain, exceptions can occur.

- If another version of the same assembly was previously loaded into the calling application domain, or if the load path of the calling application domain is different from that of the target application domain, exceptions such as MissingMethodException can occur.
- If the calling application domain makes early-bound calls to the instance type, InvalidCastException can be thrown when an attempt is made to cast the instance.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFrom(String, String, Object[])

Creates a new instance of the specified type defined in the specified assembly file.

C#

```
public System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom (string
assemblyFile, string typeName, object?[]? activationAttributes);
```

Parameters

`assemblyFile` String

The name, including the path, of a file that contains an assembly that defines the requested type. The assembly is loaded using the LoadFrom(String) method.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects.Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

ObjectHandle

An object that is a wrapper for the new instance, or `null` if `typeName` is not found. The return value needs to be unwrapped to access the real object.

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

FileNotFoundException

`assemblyFile` was not found.

TypeLoadException

`typeName` was not found in `assemblyFile`.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

MissingMethodException

No matching public constructor was found.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyFile` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

NullReferenceException

This instance is `null`.

Remarks

The parameterless constructor for `typeName` is invoked.

For more information about this method, see the Activator.CreateInstanceFrom method.

When the CreateInstanceFrom method is used to create an instance in a target application domain, other than the application domain from which the call is made, the assembly is loaded in the target application domain. However, if the instance is unwrapped in the calling application domain, using the unwrapped instance in certain ways can cause the assembly to be loaded into the calling application domain. For example, after the instance is unwrapped, its type information might be requested, in order to call its methods late-bound. When the assembly is loaded into the calling application domain, exceptions can occur.

- If another version of the same assembly was previously loaded into the calling application domain, or if the load path of the calling application domain is different from that of the target application domain, exceptions such as MissingMethodException can occur.
- If the calling application domain makes early-bound calls to the instance type, InvalidCastException can be thrown when an attempt is made to cast the instance.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFrom(String, String)

Creates a new instance of the specified type defined in the specified assembly file.

```C#
public System.Runtime.Remoting.ObjectHandle? CreateInstanceFrom (string assemblyFile, string typeName);
```

Parameters

`assemblyFile` String

The name, including the path, of a file that contains an assembly that defines the requested type. The assembly is loaded using the LoadFrom(String) method.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

Returns

ObjectHandle

An object that is a wrapper for the new instance, or `null` if `typeName` is not found. The return value needs to be unwrapped to access the real object.

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

-or-

`typeName` is `null`.

FileNotFoundException

`assemblyFile` was not found.

TypeLoadException

`typeName` was not found in `assemblyFile`.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

MissingMethodException

No parameterless public constructor was found.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyFile` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

NullReferenceException

This instance is `null`.

Examples

The following example shows how to use the CreateInstanceFrom(String, String) method overload to create an instance of an object in a target application domain and call its methods.

The example defines the `MarshalableExample` class, which can be marshaled across application domain boundaries. The example builds a path to the currently executing assembly, creates a target application domain, and uses the CreateInstanceFrom(String, String) method overload to load the example assembly into the target application domain and create an instance of `MarshalableExample`.

> **Note**
>
> The path is absolute in this example, but a relative path would also work because the **Assembly.LoadFrom** method is used to load the assembly.

After unwrapping the object handle, the example demonstrates three ways to use an object in a target application domain:

- Invoking a method with late binding, using reflection. This requires type information, which causes the assembly to be loaded into the application domain of the caller. (In this example, it is already loaded.)
- Casting the object to an interface known to both the caller and the callee. If the interface is defined in the calling assembly or in a third assembly referenced by both the caller and the callee, the called assembly is not loaded into the application domain of the caller.
- Using the object directly when its type is known to the caller. The assembly must be loaded into the application domain of the caller.

Another way to avoid loading the called assembly into the application domain of the caller is for the caller to derive from the MarshalByRefObject class and to define a method that can be run in the target application domain. That method can use reflection to examine a target assembly, because the target assembly is already loaded into the target application domain. See the example for the DynamicDirectory property.

C#

```
using System;

public interface ITest
{
    void Test(string greeting);
}

public class MarshalableExample : MarshalByRefObject, ITest
{
    static void Main()
    {
        // Construct a path to the current assembly.
        string assemblyPath = Environment.CurrentDirectory + "\\" +
            typeof(MarshalableExample).Assembly.GetName().Name + ".exe";

        AppDomain ad = AppDomain.CreateDomain("MyDomain");

        System.Runtime.Remoting.ObjectHandle oh =
            ad.CreateInstanceFrom(assemblyPath, "MarshalableExample");

        object obj = oh.Unwrap();

        // Three ways to use the newly created object, depending on how
        // much is known about the type: Late bound, early bound through
        // a mutually known interface, or early binding of a known type.
        //
        obj.GetType().InvokeMember("Test",
            System.Reflection.BindingFlags.InvokeMethod,
            Type.DefaultBinder, obj, new object[] { "Hello" });
```

```
        ITest it = (ITest) obj;
        it.Test("Hi");

        MarshalableExample ex = (MarshalableExample) obj;
        ex.Test("Goodbye");
    }

    public void Test(string greeting)
    {
        Console.WriteLine("{0} from '{1}'!", greeting,
            AppDomain.CurrentDomain.FriendlyName);
    }
}

/* This example produces the following output:

Hello from 'MyDomain'!
Hi from 'MyDomain'!
Goodbye from 'MyDomain'!
 */
```

Remarks

The parameterless constructor for `typeName` is invoked.

For more information, see the Activator.CreateInstanceFrom method.

When the CreateInstanceFrom method is used to create an instance in a target application domain, other than the application domain from which the call is made, the assembly is loaded in the target application domain. However, if the instance is unwrapped in the calling application domain, using the unwrapped instance in certain ways can cause the assembly to be loaded into the calling application domain. For example, after the instance is unwrapped, its type information might be requested, in order to call its methods late-bound. When the assembly is loaded into the calling application domain, exceptions can occur.

- If another version of the same assembly was previously loaded into the calling application domain, or if the load path of the calling application domain is different from that of the target application domain, exceptions such as MissingMethodException can occur.

- If the calling application domain makes early-bound calls to the instance type, InvalidCastException can be thrown when an attempt is made to cast the instance.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.CreateInstanceFromAndUnwrap Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new instance of a specified type defined in the specified assembly file.

Overloads

Expand table

CreateInstanceFromAndUnwrap(String, String)	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFromAndUnwrap(String, String, Object[])	Creates a new instance of the specified type defined in the specified assembly file.
CreateInstanceFromAndUnwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Creates a new instance of the specified type defined in the specified assembly file, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.

CreateInstanceFromAndUnwrap(String, String)

Creates a new instance of the specified type defined in the specified assembly file.

C#

```
public object? CreateInstanceFromAndUnwrap (string assemblyFile, string
typeName);
```

Parameters

`assemblyFile` String

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

Returns

Object

The requested object, or `null` if `typeName` is not found.

Exceptions

ArgumentNullException

`assemblyName` is `null`.

-or-

`typeName` is `null`.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileNotFoundException

`assemblyName` was not found.

TypeLoadException

`typeName` was not found in `assemblyName`.

MissingMethodException

No parameterless public constructor was found.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException
An assembly or module was loaded twice with two different evidences.

Remarks

This is a convenience method that combines CreateInstanceFrom and ObjectHandle.Unwrap. This method calls the parameterless constructor for `typeName`.

For more information, see the Activator.CreateInstanceFrom method.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFromAndUnwrap(String, String, Object[])

Creates a new instance of the specified type defined in the specified assembly file.

C#

```
public object? CreateInstanceFromAndUnwrap (string assemblyFile, string
typeName, object?[]? activationAttributes);
```

Parameters

`assemblyFile` String

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly (see the FullName property).

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects.Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

The requested object, or `null` if `typeName` is not found.

Exceptions

ArgumentNullException

`assemblyName` is `null`.

-or-

`typeName` is `null`.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileNotFoundException

`assemblyName` was not found.

TypeLoadException

`typeName` was not found in `assemblyName`.

MissingMethodException

No parameterless public constructor was found.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

This is a convenience method that combines CreateInstanceFrom and ObjectHandle.Unwrap. This method calls the parameterless constructor for `typeName`.

For more information about this method, see the Activator.CreateInstanceFrom method.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

CreateInstanceFromAndUnwrap(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])

Creates a new instance of the specified type defined in the specified assembly file, specifying whether the case of the type name is ignored; the binding attributes and the binder that are used to select the type to be created; the arguments of the constructor; the culture; and the activation attributes.

C#

```
public object? CreateInstanceFromAndUnwrap (string assemblyFile, string
typeName, bool ignoreCase, System.Reflection.BindingFlags bindingAttr,
System.Reflection.Binder? binder, object?[]? args,
System.Globalization.CultureInfo? culture, object?[]?
activationAttributes);
```

Parameters

`assemblyFile` String

The file name and path of the assembly that defines the requested type.

`typeName` String

The fully qualified name of the requested type, including the namespace but not the assembly, as returned by the FullName property.

`ignoreCase` Boolean

A Boolean value specifying whether to perform a case-sensitive search or not.

`bindingAttr` BindingFlags

A combination of zero or more bit flags that affect the search for the `typeName` constructor. If `bindingAttr` is zero, a case-sensitive search for public constructors is conducted.

`binder` Binder

An object that enables the binding, coercion of argument types, invocation of members, and retrieval of MemberInfo objects through reflection. If `binder` is null, the default binder is used.

`args` Object[]

The arguments to pass to the constructor. This array of arguments must match in number, order, and type the parameters of the constructor to invoke. If the parameterless constructor is preferred, `args` must be an empty array or null.

`culture` CultureInfo

Culture-specific information that governs the coercion of `args` to the formal types declared for the `typeName` constructor. If `culture` is `null`, the CultureInfo for the current thread is used.

`activationAttributes` Object[]

An array of one or more attributes that can participate in activation. Typically, an array that contains a single UrlAttribute object that specifies the URL that is required to activate a remote object.

This parameter is related to client-activated objects. Client activation is a legacy technology that is retained for backward compatibility but is not recommended for new development. Distributed applications should instead use Windows Communication Foundation.

Returns

Object

The requested object, or `null` if `typeName` is not found.

Exceptions

ArgumentNullException

`assemblyName` is `null`.

-or-

`typeName` is `null`.

NotSupportedException

The caller cannot provide activation attributes for an object that does not inherit from MarshalByRefObject.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileNotFoundException

`assemblyName` was not found.

TypeLoadException

`typeName` was not found in `assemblyName`.

MissingMethodException

No matching public constructor was found.

MethodAccessException

The caller does not have sufficient permission to call this constructor.

BadImageFormatException

`assemblyName` is not a valid assembly.

-or-

`assemblyName` was compiled with a later version of the common language runtime that the version that is currently loaded.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

This is a convenience method that combines CreateInstanceFrom and ObjectHandle.Unwrap.

For more information about this method, see the Activator.CreateInstanceFrom method.

See also

- CreateInstanceFrom(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.ExecuteAssembly Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Executes the assembly contained in the specified file.

Overloads

Expand table

ExecuteAssembly(String, String[], Byte[], AssemblyHashAlgorithm)	**Obsolete.** Executes the assembly contained in the specified file, using the specified arguments, hash value, and hash algorithm.
ExecuteAssembly(String, String[])	Executes the assembly contained in the specified file, using the specified arguments.
ExecuteAssembly(String)	Executes the assembly contained in the specified file.

ExecuteAssembly(String, String[], Byte[], AssemblyHashAlgorithm)

Caution

Code Access Security is not supported or honored by the runtime.

Executes the assembly contained in the specified file, using the specified arguments, hash value, and hash algorithm.

C#

```
[System.Obsolete("Code Access Security is not supported or honored by the runtime.", DiagnosticId="SYSLIB0003", UrlFormat="https://aka.ms/dotnet-
```

```
warnings/{0}")]
public int ExecuteAssembly (string assemblyFile, string?[]? args, byte[]?
hashValue, System.Configuration.Assemblies.AssemblyHashAlgorithm
hashAlgorithm);
```

Parameters

`assemblyFile` String

The name of the file that contains the assembly to execute.

`args` String[]

The arguments to the entry point of the assembly.

`hashValue` Byte[]

Represents the value of the computed hash code.

`hashAlgorithm` AssemblyHashAlgorithm

Represents the hash algorithm used by the assembly manifest.

Returns

Int32

The value that is returned by the entry point of the assembly.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

FileNotFoundException

`assemblyFile` is not found.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

`assemblyFile` was compiled with a later version of the common language runtime than the version that is currently loaded.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

MissingMethodException

The specified assembly has no entry point.

Examples

The following sample demonstrates using one of the overloads of ExecuteAssembly on two different domains.

C#

```
class ExecuteAssemblySnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      AppDomain otherDomain = AppDomain.CreateDomain("otherDomain");

      currentDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on [default]"

      otherDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on otherDomain"
   }
}
```

Remarks

⊗ Caution

Code Access Security (CAS) has been deprecated across all versions of .NET Framework and .NET. Recent versions of .NET do not honor CAS annotations and produce errors if CAS-related APIs are used. Developers should seek alternative means of accomplishing security tasks.

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

This method loads assemblies using the LoadFile method. You can also execute assemblies using the ExecuteAssemblyByName method, which loads assemblies using the Load method.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecuteAssembly(String, String[])

Executes the assembly contained in the specified file, using the specified arguments.

C#

```
public int ExecuteAssembly (string assemblyFile, string?[]? args);
```

Parameters

`assemblyFile` String

The name of the file that contains the assembly to execute.

`args` String[]

The arguments to the entry point of the assembly.

Returns

Int32

The value that is returned by the entry point of the assembly.

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

FileNotFoundException

`assemblyFile` is not found.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

`assemblyFile` was compiled with a later version of the common language runtime than the version that is currently loaded.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

MissingMethodException

The specified assembly has no entry point.

Examples

The following sample demonstrates using one of the overloads of ExecuteAssembly on two different domains.

C#

```
class ExecuteAssemblySnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      AppDomain otherDomain = AppDomain.CreateDomain("otherDomain");

      currentDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on [default]"

      otherDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on otherDomain"
   }
}
```

Remarks

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

This method loads assemblies using the LoadFile method. You can also execute assemblies using the ExecuteAssemblyByName method, which loads assemblies using the Load method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecuteAssembly(String)

Executes the assembly contained in the specified file.

C#

```
public int ExecuteAssembly (string assemblyFile);
```

Parameters

`assemblyFile` String

The name of the file that contains the assembly to execute.

Returns

Int32

The value returned by the entry point of the assembly.

Exceptions

ArgumentNullException

`assemblyFile` is `null`.

FileNotFoundException

`assemblyFile` is not found.

BadImageFormatException

`assemblyFile` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyFile` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

MissingMethodException

The specified assembly has no entry point.

Examples

The following sample demonstrates using one of the overloads of ExecuteAssembly on two different domains.

C#

```
class ExecuteAssemblySnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      AppDomain otherDomain = AppDomain.CreateDomain("otherDomain");

      currentDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on [default]"

      otherDomain.ExecuteAssembly("MyExecutable.exe");
      // Prints "MyExecutable running on otherDomain"
   }
}
```

Remarks

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

This method loads assemblies using the LoadFile method. You can also execute assemblies using the ExecuteAssemblyByName method, which loads assemblies using the Load method.

To create the AppDomain to load and execute, use the CreateDomain method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ExecuteAssemblyByName Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Executes an assembly.

Overloads

Expand table

ExecuteAssemblyByName(String)	Executes an assembly given its display name.
ExecuteAssemblyByName(AssemblyName, String[])	Executes the assembly given an AssemblyName, using the specified arguments.
ExecuteAssemblyByName(String, String[])	Executes the assembly given its display name, using the specified arguments.

ExecuteAssemblyByName(String)

Executes an assembly given its display name.

C#

```
public int ExecuteAssemblyByName (string assemblyName);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

Returns

Int32

The value returned by the entry point of the assembly.

Exceptions

ArgumentNullException

`assemblyName` is `null`.

FileNotFoundException

The assembly specified by `assemblyName` is not found.

BadImageFormatException

The assembly specified by `assemblyName` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyName` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

The assembly specified by `assemblyName` was found, but could not be loaded.

MissingMethodException

The specified assembly has no entry point.

Remarks

The ExecuteAssemblyByName method provides similar functionality to the ExecuteAssembly method, but specifies the assembly by display name or AssemblyName rather than by file location. Therefore, ExecuteAssemblyByName loads assemblies with the Load method rather than with the LoadFile method.

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

To create the AppDomain to load and execute, use the CreateDomain method.

See also

- AssemblyName
- Load(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecuteAssemblyByName(AssemblyName, String[])

Executes the assembly given an AssemblyName, using the specified arguments.

C#

```
public int ExecuteAssemblyByName (System.Reflection.AssemblyName
assemblyName, params string?[]? args);
```

Parameters

`assemblyName` AssemblyName

An AssemblyName object representing the name of the assembly.

`args` String[]

Command-line arguments to pass when starting the process.

Returns

Int32

The value that is returned by the entry point of the assembly.

Exceptions

FileNotFoundException

The assembly specified by `assemblyName` is not found.

FileLoadException

The assembly specified by `assemblyName` was found, but could not be loaded.

BadImageFormatException

The assembly specified by `assemblyName` is not a valid assembly.

-or-

`assemblyName` was compiled with a later version of the common language runtime than the version that is currently loaded.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

MissingMethodException

The specified assembly has no entry point.

Remarks

The ExecuteAssemblyByName method provides similar functionality to the ExecuteAssembly method, but specifies the assembly by display name or AssemblyName rather than by file location. Therefore, ExecuteAssemblyByName loads assemblies with the Load method rather than with the LoadFile method.

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecuteAssemblyByName(String, String[])

Executes the assembly given its display name, using the specified arguments.

```
C#

public int ExecuteAssemblyByName (string assemblyName, params string?[]?
args);
```

Parameters

`assemblyName` String

The display name of the assembly. See FullName.

`args` String[]

Command-line arguments to pass when starting the process.

Returns

Int32

The value that is returned by the entry point of the assembly.

Exceptions

ArgumentNullException

`assemblyName` is `null`.

FileNotFoundException

The assembly specified by `assemblyName` is not found.

FileLoadException

The assembly specified by `assemblyName` was found, but could not be loaded.

BadImageFormatException

The assembly specified by `assemblyName` is not a valid assembly.

-or-

`assemblyName` was compiled with a later version of the common language runtime than the version that is currently loaded.

AppDomainUnloadedException
The operation is attempted on an unloaded application domain.

MissingMethodException
The specified assembly has no entry point.

Remarks

The ExecuteAssemblyByName method provides similar functionality to the ExecuteAssembly method, but specifies the assembly by display name or AssemblyName rather than by file location. Therefore, ExecuteAssemblyByName loads assemblies with the Load method rather than with the LoadFile method.

The assembly begins executing at the entry point specified in the .NET Framework header.

This method does not create a new process or application domain, and it does not execute the entry point method on a new thread.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

AppDomain.GetAssemblies Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the assemblies that have been loaded into the execution context of this application domain.

C#

```
public System.Reflection.Assembly[] GetAssemblies ();
```

Returns

Assembly[]

An array of assemblies in this application domain.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following code example uses the GetAssemblies method to get a list of all assemblies that have been loaded into the application domain. The assemblies are then displayed to the console.

To run this code example, you need to create an assembly named `CustomLibrary.dll`, or change the assembly name that is passed to the GetAssemblies method.

C#

```
using System;
using System.Reflection;
using System.Security.Policy;

class ADGetAssemblies
```

```
{

    public static void Main()
    {
        AppDomain currentDomain = AppDomain.CurrentDomain;
        //Provide the current application domain evidence for the assembly.
        Evidence asEvidence = currentDomain.Evidence;
        //Load the assembly from the application directory using a simple
name.

        //Create an assembly called CustomLibrary to run this sample.
        currentDomain.Load("CustomLibrary",asEvidence);

        //Make an array for the list of assemblies.
        Assembly[] assems = currentDomain.GetAssemblies();

        //List the assemblies in the current application domain.
        Console.WriteLine("List of assemblies loaded in current
appdomain:");
            foreach (Assembly assem in assems)
                Console.WriteLine(assem.ToString());
    }
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.GetCurrentThreadId Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.GetCurrentThreadId has been deprecated because it does not provide a stable Id when managed threads are running on fibers (aka lightweight threads). To get a stable identifier for a managed thread, use the ManagedThreadId property on Thread instead.

Gets the current thread identifier.

C#

```
[System.Obsolete("AppDomain.GetCurrentThreadId has been deprecated because it does not provide a stable Id when managed threads are running on fibers (aka lightweight threads). To get a stable identifier for a managed thread, use the ManagedThreadId property on Thread instead.")]
public static int GetCurrentThreadId ();
```

Returns

Int32

A 32-bit signed integer that is the identifier of the current thread.

Attributes ObsoleteAttribute

Remarks

Use the Thread.ManagedThreadId property, which is stable even when the .NET Framework is hosted by an environment that supports fibers (that is, lightweight

threads).

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- Int32
- ManagedThreadId

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.GetData(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value stored in the current application domain for the specified name.

```
C#
public object? GetData (string name);
```

Parameters

name String

The name of a predefined application domain property, or the name of an application domain property you have defined.

Returns

Object

The value of the `name` property, or `null` if the property does not exist.

Exceptions

ArgumentNullException

`name` is `null`.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following example creates a new application domain, sets a system-provided value for the domain, and adds a new value pair for the domain. The example then demonstrates how to use the GetData method to retrieve the data from these value pairs and display them to the console.

C#

```
using System;
using System.Reflection;

class ADGetData
{
    public static void Main()
    {
        // appdomain setup information
        AppDomain currentDomain = AppDomain.CurrentDomain;

        //Create a new value pair for the appdomain
        currentDomain.SetData("ADVALUE", "Example value");

        //get the value specified in the setdata method
        Console.WriteLine("ADVALUE is: " +
currentDomain.GetData("ADVALUE"));

        //get a system value specified at appdomainsetup
        Console.WriteLine("System value for loader optimization: {0}",
            currentDomain.GetData("LOADER_OPTIMIZATION"));
    }
}

/* This code example produces the following output:

ADVALUE is: Example value
System value for loader optimization: NotSpecified
 */
```

Remarks

Use this method to retrieve the value of an entry in an internal cache of name-data pairs that describe properties of this instance of AppDomain. Note that the comparison of `name` with the name of key-value pairs is case-sensitive.

The cache automatically contains predefined system entries that are inserted when the application domain is created. You can inspect their values with the GetData method, or the equivalent AppDomainSetup properties.

You can insert or modify your own user defined name-data pairs with the SetData method and inspect their values with the GetData method.

The following table describes the `name` of each predefined system entry and its corresponding AppDomainSetup property.

Expand table

Value of 'name'	Property
"APPBASE"	AppDomainSetup.ApplicationBase
"APP_CONFIG_FILE"	AppDomainSetup.ConfigurationFile
"APP_LAUNCH_URL"	(no property) "APP_LAUNCH_URL" represents the URL originally requested by the user, before any redirection. It is available only when the application has been launched with a browser. Not all browsers provide this value.
"APP_NAME"	AppDomainSetup.ApplicationName
"BINPATH_PROBE_ONLY"	AppDomainSetup.PrivateBinPathProbe
"CACHE_BASE"	AppDomainSetup.CachePath
"CODE_DOWNLOAD_DISABLED"	AppDomainSetup.DisallowCodeDownload
"DEV_PATH"	(no property)
"DISALLOW_APP"	AppDomainSetup.DisallowPublisherPolicy
"DISALLOW_APP_BASE_PROBING"	AppDomainSetup.DisallowApplicationBaseProbing
"DISALLOW_APP_REDIRECTS"	AppDomainSetup.DisallowBindingRedirects
"DYNAMIC_BASE"	AppDomainSetup.DynamicBase
"FORCE_CACHE_INSTALL"	AppDomainSetup.ShadowCopyFiles
"LICENSE_FILE", or an application-specific string	AppDomainSetup.LicenseFile
"LOADER_OPTIMIZATION"	AppDomainSetup.LoaderOptimization
"LOCATION_URI"	(no property)
"PRIVATE_BINPATH"	AppDomainSetup.PrivateBinPath
"REGEX_DEFAULT_MATCH_TIMEOUT"	Regex.MatchTimeout "REGEX_DEFAULT_MATCH_TIMEOUT" is not a system entry, and its value can be set by calling the SetData method.
"SHADOW_COPY_DIRS"	AppDomainSetup.ShadowCopyDirectories

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SetData(String, Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.IsCompatibilitySwitchSet(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a nullable Boolean value that indicates whether any compatibility switches are set, and if so, whether the specified compatibility switch is set.

C#

```
public bool? IsCompatibilitySwitchSet (string value);
```

Parameters

`value` String

The compatibility switch to test.

Returns

Nullable<Boolean>

A null reference (`Nothing` in Visual Basic) if no compatibility switches are set; otherwise, a Boolean value that indicates whether the compatibility switch that is specified by `value` is set.

Remarks

This method tests whether the specified compatibility switch has been set for the current application domain. Compatibility switches typically restore a behavior (such as the way strings are sorted) that was changed between versions of the .NET Framework. They are set by calling the AppDomainSetup.SetCompatibilitySwitches method before creating an application domain.

The following table provides examples of compatibility switches that can be set to restore the behavior of earlier versions of the .NET Framework.

Expand table

Switch	Meaning
"NetFx40_LegacySecurityPolicy"	Code access security (CAS) for the .NET Framework 3.5 is enabled in this application domain. See <NetFx40_LegacySecurityPolicy> Element.
"NetFx40_Legacy20SortingBehavior"	String sorting defaults for the .NET Framework 3.5 are enabled in this application domain. Its success requires sort00001000.dll to be installed. See <CompatSortNLSVersion> Element.
"NetFx40_Legacy40SortingBehavior"	String sorting defaults for the .NET Framework 4and Unicode 5.0 are enabled in this application domain. Its success requires sort00060101.dll to be installed.
"NetFx40_TimeSpanLegacyFormatMode"	TimeSpan formatting behavior for the .NET Framework 3.5 is enabled in this application domain. See <TimeSpan_LegacyFormatMode> Element and the "Restoring Legacy TimeSpan Formatting" section of the TimeSpan topic.
"UseRandomizedStringHashAlgorithm"	The runtime calculates hash codes for strings on a per application domain basis instead of using a single hashing algorithm that produces a consistent hash code across application domains. See <UseRandomizedStringHashAlgorithm> Element.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SetCompatibilitySwitches(IEnumerable<String>)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.IsDefaultAppDomain Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the application domain is the default application domain for the process.

C#

```
public bool IsDefaultAppDomain ();
```

Returns

Boolean

`true` if the current AppDomain object represents the default application domain for the process; otherwise, `false`.

Examples

The following code example creates a second application domain and displays information about the default domain and the new domain.

C#

```
using System;
using System.Reflection;

public class AppDomainIdSnippet
{
    // The following attribute indicates to the loader that assemblies
    // in the global assembly cache should be shared across multiple
    // application domains.
    [LoaderOptimizationAttribute( LoaderOptimization.MultiDomainHost)]
    public static void Main()
    {
        // Show information for the default application domain.
```

```
        ShowDomainInfo();

        // Create a new application domain and display its information.
        AppDomain newDomain = AppDomain.CreateDomain("MyMultiDomain");
        newDomain.DoCallBack(new CrossAppDomainDelegate(ShowDomainInfo));
    }

    // This method has the same signature as the CrossAppDomainDelegate,
    // so that it can be executed easily in the new application domain.
    //
    public static void ShowDomainInfo()
    {
        AppDomain ad = AppDomain.CurrentDomain;
        Console.WriteLine();
        Console.WriteLine("FriendlyName: {0}", ad.FriendlyName);
        Console.WriteLine("Id: {0}", ad.Id);
        Console.WriteLine("IsDefaultAppDomain: {0}",
ad.IsDefaultAppDomain());
    }
}
```

Remarks

Every managed process has a default application domain. Execution begins in the default domain.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

AppDomain.IsFinalizingForUnload Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether this application domain is unloading, and the objects it contains are being finalized by the common language runtime.

C#

```
public bool IsFinalizingForUnload ();
```

Returns

Boolean

`true` if this application domain is unloading and the common language runtime has started invoking finalizers; otherwise, `false`.

Remarks

The finalization method for an object provides you an opportunity to perform any necessary cleanup operations before the object is garbage collected. After finalization, the object is accessible but in an invalid state and therefore unusable. Eventually, garbage collection completes and reclaims the object.

An object's finalization method is called in one of the following situations: during garbage collection, when the common language runtime is shutting down, or when the application domain that contains the object is unloaded. The IsFinalizingForUnload method returns `true` only in the last case; it does not return `true` if finalization results from routine garbage collection or from CLR shutdown.

Note

To determine whether finalization is due to CLR shutdown, use the **Environment.HasShutdownStarted** property. It returns `true` if finalization is due to an application domain being unloaded or to the CLR shutting down.

While executing in your finalization method during domain unloading, you might want to access another object that is referenced by a static field and has a finalization method. However, you cannot reliably do so because the accessed object might already have been finalized.

> **Note**
>
> An exception to this rule is the **Console** class, which contains static fields that reference stream objects, but is implemented specially so you can always write to the system console, even during domain unloading or system shutdown.

Use this method in an object's finalization method to determine whether the application domain that contains the object is unloading. If that is the case, you cannot reliably access any object that has a finalization method and is referenced by a static field.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- HasShutdownStarted

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

AppDomain.Load Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Loads an Assembly into this application domain.

Overloads

Expand table

Load(Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly.
Load(AssemblyName)	Loads an Assembly given its AssemblyName.
Load(String)	Loads an Assembly given its display name.
Load(Byte[], Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly. The raw bytes representing the symbols for the Assembly are also loaded.

Load(Byte[])

Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly.

C#

```
public System.Reflection.Assembly Load (byte[] rawAssembly);
```

Parameters

rawAssembly Byte[]

An array of type `byte` that is a COFF-based image containing an emitted assembly.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`rawAssembly` is `null`.

BadImageFormatException

`rawAssembly` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `rawAssembly` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following sample demonstrates the use of loading a raw assembly.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.IO;
using System.Reflection;
using System.Reflection.Emit;

class LoadRawSnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;

      InstantiateMyType(currentDomain);   // Failed!
```

```
      currentDomain.AssemblyResolve += new
ResolveEventHandler(MyResolver);

      InstantiateMyType(currentDomain);   // OK!
   }

   static void InstantiateMyType(AppDomain domain) {
      try {
     // You must supply a valid fully qualified assembly name here.
         domain.CreateInstance("Assembly text name, Version, Culture,
PublicKeyToken", "MyType");
      } catch (Exception e) {
         Console.WriteLine(e.Message);
      }
   }

   // Loads the content of a file to a byte array.
   static byte[] loadFile(string filename) {
      FileStream fs = new FileStream(filename, FileMode.Open);
      byte[] buffer = new byte[(int) fs.Length];
      fs.Read(buffer, 0, buffer.Length);
      fs.Close();

      return buffer;
   }

   static Assembly MyResolver(object sender, ResolveEventArgs args) {
      AppDomain domain = (AppDomain) sender;

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly(domain);

      byte[] rawAssembly = loadFile("temp.dll");
      byte[] rawSymbolStore = loadFile("temp.pdb");
      Assembly assembly = domain.Load(rawAssembly, rawSymbolStore);

      return assembly;
   }

   // Creates a dynamic assembly with symbol information
   // and saves them to temp.dll and temp.pdb
   static void EmitAssembly(AppDomain domain) {
      AssemblyName assemblyName = new AssemblyName();
      assemblyName.Name = "MyAssembly";

      AssemblyBuilder assemblyBuilder =
domain.DefineDynamicAssembly(assemblyName, AssemblyBuilderAccess.Save);
      ModuleBuilder moduleBuilder =
assemblyBuilder.DefineDynamicModule("MyModule", "temp.dll", true);
      TypeBuilder typeBuilder = moduleBuilder.DefineType("MyType",
TypeAttributes.Public);

      ConstructorBuilder constructorBuilder =
typeBuilder.DefineConstructor(MethodAttributes.Public,
```

```
CallingConventions.Standard, null);
      ILGenerator ilGenerator = constructorBuilder.GetILGenerator();
      ilGenerator.EmitWriteLine("MyType instantiated!");
      ilGenerator.Emit(OpCodes.Ret);

      typeBuilder.CreateType();

      assemblyBuilder.Save("temp.dll");
   }
}
```

Remarks

Beginning with the .NET Framework 4, the trust level of an assembly that is loaded by using this method is the same as the trust level of the application domain.

This method should be used only to load an assembly into the current application domain. This method is provided as a convenience for interoperability callers who cannot call the static Assembly.Load method. To load assemblies into other application domains, use a method such as CreateInstanceAndUnwrap.

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Load(AssemblyName)

Loads an Assembly given its AssemblyName.

C#

```
public System.Reflection.Assembly Load (System.Reflection.AssemblyName
```

```
assemblyRef);
```

Parameters

`assemblyRef` AssemblyName

An object that describes the assembly to load.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`assemblyRef` is `null`.

FileNotFoundException

`assemblyRef` is not found.

BadImageFormatException

`assemblyRef` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyRef` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

This method should be used only to load an assembly into the current application domain. This method is provided as a convenience for interoperability callers who cannot call the static Assembly.Load method. To load assemblies into other application domains, use a method such as CreateInstanceAndUnwrap.

If a version of the requested assembly is already loaded, this method returns the loaded assembly, even if a different version is requested.

Supplying a partial assembly name for `assemblyRef` is not recommended. (A partial name omits one or more of culture, version, or public key token. For overloads that take a string instead of an AssemblyName object, "MyAssembly, Version=1.0.0.0" is an example of a partial name and "MyAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=18ab3442da84b47" is an example of a full name.) Using partial names has a negative effect on performance. In addition, a partial assembly name can load an assembly from the global assembly cache only if there is an exact copy of the assembly in the application base directory (BaseDirectory or AppDomainSetup.ApplicationBase).

If the current AppDomain object represents application domain `A`, and the Load method is called from application domain `B`, the assembly is loaded into both application domains. For example, the following code loads `MyAssembly` into the new application domain `ChildDomain` and also into the application domain where the code executes:

```
C#

AppDomain ad = AppDomain.CreateDomain("ChildDomain");
ad.Load("MyAssembly");
```

The assembly is loaded into both domains because Assembly does not derive from MarshalByRefObject, and therefore the return value of the Load method cannot be marshaled. Instead, the common language runtime tries to load the assembly into the calling application domain. The assemblies that are loaded into the two application domains might be different if the path settings for the two application domains are different.

> **Note**
>
> If both the **AssemblyName.Name** property and the **AssemblyName.CodeBase** property are set, the first attempt to load the assembly uses the display name (including version, culture, and so on, as returned by the **Assembly.FullName** property). If the file is not found, the **CodeBase** property is used to search for the assembly. If the assembly is found using **CodeBase**, the display name is matched against the assembly. If the match fails, a **FileLoadException** is thrown.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Load(String)

Loads an Assembly given its display name.

```
C#

public System.Reflection.Assembly Load (string assemblyString);
```

Parameters

`assemblyString` String

The display name of the assembly. See FullName.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`assemblyString` is `null`

FileNotFoundException

`assemblyString` is not found.

BadImageFormatException

`assemblyString` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `assemblyString` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

This method should be used only to load an assembly into the current application domain. This method is provided as a convenience for interoperability callers who cannot call the static Assembly.Load method. To load assemblies into other application domains, use a method such as CreateInstanceAndUnwrap.

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Load(Byte[], Byte[])

Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly. The raw bytes representing the symbols for the Assembly are also loaded.

C#

```
public System.Reflection.Assembly Load (byte[] rawAssembly, byte[]?
rawSymbolStore);
```

Parameters

`rawAssembly` Byte[]

An array of type `byte` that is a COFF-based image containing an emitted assembly.

`rawSymbolStore` Byte[]

An array of type `byte` containing the raw bytes representing the symbols for the assembly.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`rawAssembly` is `null`.

BadImageFormatException

`rawAssembly` is not a valid assembly.

-or-

Version 2.0 or later of the common language runtime is currently loaded and `rawAssembly` was compiled with a later version.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following sample demonstrates the use of loading a raw assembly.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

```
C#
```

```
using System;
using System.IO;
using System.Reflection;
using System.Reflection.Emit;

class LoadRawSnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;

      InstantiateMyType(currentDomain);   // Failed!

      currentDomain.AssemblyResolve += new
ResolveEventHandler(MyResolver);

      InstantiateMyType(currentDomain);   // OK!
   }

   static void InstantiateMyType(AppDomain domain) {
      try {
     // You must supply a valid fully qualified assembly name here.
         domain.CreateInstance("Assembly text name, Version, Culture,
PublicKeyToken", "MyType");
      } catch (Exception e) {
         Console.WriteLine(e.Message);
      }
   }

   // Loads the content of a file to a byte array.
   static byte[] loadFile(string filename) {
      FileStream fs = new FileStream(filename, FileMode.Open);
      byte[] buffer = new byte[(int) fs.Length];
      fs.Read(buffer, 0, buffer.Length);
      fs.Close();

      return buffer;
   }

   static Assembly MyResolver(object sender, ResolveEventArgs args) {
      AppDomain domain = (AppDomain) sender;

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly(domain);

      byte[] rawAssembly = loadFile("temp.dll");
      byte[] rawSymbolStore = loadFile("temp.pdb");
      Assembly assembly = domain.Load(rawAssembly, rawSymbolStore);

      return assembly;
   }

   // Creates a dynamic assembly with symbol information
   // and saves them to temp.dll and temp.pdb
   static void EmitAssembly(AppDomain domain) {
```

```
        AssemblyName assemblyName = new AssemblyName();
        assemblyName.Name = "MyAssembly";

        AssemblyBuilder assemblyBuilder =
domain.DefineDynamicAssembly(assemblyName, AssemblyBuilderAccess.Save);
        ModuleBuilder moduleBuilder =
assemblyBuilder.DefineDynamicModule("MyModule", "temp.dll", true);
        TypeBuilder typeBuilder = moduleBuilder.DefineType("MyType",
TypeAttributes.Public);

        ConstructorBuilder constructorBuilder =
typeBuilder.DefineConstructor(MethodAttributes.Public,
CallingConventions.Standard, null);
        ILGenerator ilGenerator = constructorBuilder.GetILGenerator();
        ilGenerator.EmitWriteLine("MyType instantiated!");
        ilGenerator.Emit(OpCodes.Ret);

        typeBuilder.CreateType();

        assemblyBuilder.Save("temp.dll");
    }
}
```

Remarks

Beginning with the .NET Framework 4, the trust level of an assembly that is loaded by using this method is the same as the trust level of the application domain.

This method should be used only to load an assembly into the current application domain. This method is provided as a convenience for interoperability callers who cannot call the static Assembly.Load method. To load assemblies into other application domains, use a method such as CreateInstanceAndUnwrap.

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.ReflectionOnlyGetAssemblies Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the assemblies that have been loaded into the reflection-only context of the application domain.

C#

```
public System.Reflection.Assembly[] ReflectionOnlyGetAssemblies ();
```

Returns

Assembly[]

An array of Assembly objects that represent the assemblies loaded into the reflection-only context of the application domain.

Exceptions

AppDomainUnloadedException

An operation is attempted on an unloaded application domain.

Examples

The following code example loads the System.dll assembly into the execution context and then into the reflection-only context. The GetAssemblies and ReflectionOnlyGetAssemblies methods are used to display the assemblies loaded into each context.

C#

```
using System;
using System.Reflection;
using System.Timers;
```

```
public class ReflectionOnlySnippet
{
    public static void Main()
    {
        // Get the assembly display name for System.dll, the assembly
        // that contains System.Timers.Timer. Note that this causes
        // System.dll to be loaded into the execution context.
        //
        string displayName = typeof(Timer).Assembly.FullName;

        // Load System.dll into the reflection-only context. Note that
        // if you obtain the display name (for example, by running this
        // example program), and enter it as a literal string in the
        // preceding line of code, you can load System.dll into the
        // reflection-only context without loading it into the execution
        // context.
        Assembly.ReflectionOnlyLoad(displayName);

        // Display the assemblies loaded into the execution and
        // reflection-only contexts. System.dll appears in both contexts.
        //
        Console.WriteLine("------------- Execution Context --------------");
        foreach (Assembly a in AppDomain.CurrentDomain.GetAssemblies())
        {
            Console.WriteLine("\t{0}", a.GetName());
        }
        Console.WriteLine("------------- Reflection-only Context -----------
---");
        foreach (Assembly a in
AppDomain.CurrentDomain.ReflectionOnlyGetAssemblies())
        {
            Console.WriteLine("\t{0}", a.GetName());
        }
    }
}
```

Remarks

This method returns the assemblies that have been loaded into the reflection-only context. To get the assemblies that have been loaded for execution, use the GetAssemblies method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetAssemblies()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetCachePath(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.SetCachePath has been deprecated and is not supported.

Establishes the specified directory path as the location where assemblies are shadow copied.

C#

```
[System.Obsolete("AppDomain.SetCachePath has been deprecated and is not supported.")]
public void SetCachePath (string? path);
```

Parameters

path String

The fully qualified path to the shadow copy location.

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Remarks

The cache path is ignored if the ApplicationName property is not set. See the AppDomainSetup.CachePath property.

For more information on shadow copying see Shadow Copying Assemblies.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- CachePath
- Shadow Copying Assemblies

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Assigns a value to an application domain property.

Overloads

Expand table

SetData(String, Object)	Assigns the specified value to the specified application domain property.

SetData(String, Object)

Assigns the specified value to the specified application domain property.

C#

```
public void SetData (string name, object? data);
```

Parameters

`name` String

The name of a user-defined application domain property to create or change.

`data` Object

The value of the property.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following example demonstrates how to use the SetData(String, Object) method to create a new value pair. The example then uses the GetData method to retrieve the value, and displays it to the console.

C#

```
using System;
using System.Reflection;

class ADGetData
{
    public static void Main()
    {
        // appdomain setup information
        AppDomain currentDomain = AppDomain.CurrentDomain;

        //Create a new value pair for the appdomain
        currentDomain.SetData("ADVALUE", "Example value");

        //get the value specified in the setdata method
        Console.WriteLine("ADVALUE is: " +
currentDomain.GetData("ADVALUE"));

        //get a system value specified at appdomainsetup
        Console.WriteLine("System value for loader optimization: {0}",
            currentDomain.GetData("LOADER_OPTIMIZATION"));
    }
}

/* This code example produces the following output:

ADVALUE is: Example value
System value for loader optimization: NotSpecified
 */
```

Remarks

Use this method to insert an entry, or modify the value of an entry in an internal cache of name-data pairs that describe properties of this instance of AppDomain.

The cache automatically contains predefined system entries that are inserted when the application domain is created. You cannot insert or modify system entries with this method. A method call that attempts to modify a system entry has no effect; the method does not throw an exception. You can inspect the values of system entries with the GetData method, or the equivalent AppDomainSetup properties described in GetData.

You can call this method to set the value of the default timeout interval for evaluating regular expression patterns by supply "REGEX_DEFAULT_MATCH_TIMEOUT" as the value of the `name` argument and a TimeSpan value that represents the timeout interval as the value of the `data` argument. You can also insert or modify your own user defined name-data pairs with this method and inspect their values with the GetData method.

See also

- GetData(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetDynamicBase(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.SetDynamicBase has been deprecated and is not supported.

Establishes the specified directory path as the base directory for subdirectories where dynamically generated files are stored and accessed.

C#

```
[System.Obsolete("AppDomain.SetDynamicBase has been deprecated and is not supported.")]
public void SetDynamicBase (string? path);
```

Parameters

path String

The fully qualified path that is the base directory for subdirectories where dynamic assemblies are stored.

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

This method is now obsolete, and should not be used for new development. The following example shows how to use the non-obsolete alternative, the AppDomainSetup.DynamicBase property. For an explanation of this example, see the AppDomainSetup.DynamicBase property or the DynamicDirectory property.

C#

```
using System;
using System.Reflection;
using System.Reflection.Emit;

public class AddDynamicBaseSnippet : MarshalByRefObject
{
   static void Main()
   {
      // Prepare to create a new application domain.
      AppDomainSetup setup = new AppDomainSetup();

      // Set the application name before setting the dynamic base.
      setup.ApplicationName = "Example";

      // Set the location of the base directory where assembly resolution
      // probes for dynamic assemblies. Note that the hash code of the
      // application name is concatenated to the base directory name you
      // supply.
      setup.DynamicBase = "C:\\DynamicAssemblyDir";
      Console.WriteLine("DynamicBase is set to '{0}'.", setup.DynamicBase);

      AppDomain ad = AppDomain.CreateDomain("MyDomain", null, setup);

      // The dynamic directory name is the dynamic base concatenated with
      // the application name: <DynamicBase>\<hash code>\<ApplicationName>
      string dynamicDir = ad.DynamicDirectory;
      Console.WriteLine("Dynamic directory is '{0}'.", dynamicDir);

      // The AssemblyBuilder won't create this directory automatically.
      if (!System.IO.Directory.Exists(dynamicDir))
      {
         Console.WriteLine("Creating the dynamic directory.");
         System.IO.Directory.CreateDirectory(dynamicDir);
      }

      // Generate a dynamic assembly and store it in the dynamic
      // directory.
      GenerateDynamicAssembly(dynamicDir);

      // Create an instance of the Example class in the application domain,
      // and call its Test method to load the dynamic assembly and use it.
      AddDynamicBaseSnippet ex = (AddDynamicBaseSnippet)
ad.CreateInstanceAndUnwrap(
         typeof(AddDynamicBaseSnippet).Assembly.FullName, "Example");
      ex.Test();
   }
```

```
   public void Test()
   {
      Assembly dynAssem = Assembly.Load(
         "DynamicHelloWorld, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null");

      Type myType = dynAssem.GetType("HelloWorld");
      myType.InvokeMember("HelloFromAD", BindingFlags.Public |
         BindingFlags.Static | BindingFlags.InvokeMethod,
         Type.DefaultBinder, null, null);
   }

   private static void GenerateDynamicAssembly(string location)
   {
      // Define the dynamic assembly and the module. There is only one
      // module in this assembly. Note that the call to
DefineDynamicAssembly
      // specifies the location where the assembly will be saved. The
      // assembly version is 1.0.0.0.
      //
      AssemblyName asmName = new AssemblyName("DynamicHelloWorld");
      asmName.Version = new Version("1.0.0.0");

      AssemblyBuilder ab =
         AppDomain.CurrentDomain.DefineDynamicAssembly(
            asmName, AssemblyBuilderAccess.Save, location);

      String moduleName = asmName.Name + ".exe";
      ModuleBuilder mb = ab.DefineDynamicModule(asmName.Name, moduleName);

      // Define the "HelloWorld" type, with one static method.
      TypeBuilder tb = mb.DefineType("HelloWorld", TypeAttributes.Public);
      MethodBuilder hello = tb.DefineMethod("HelloFromAD",
         MethodAttributes.Public | MethodAttributes.Static, null, null);

      // The method displays a message that contains the name of the
application
      // domain where the method is executed.
      ILGenerator il = hello.GetILGenerator();
      il.Emit(OpCodes.Ldstr, "Hello from '{0}'!");
      il.Emit(OpCodes.Call,
typeof(AppDomain).GetProperty("CurrentDomain").GetGetMethod());
      il.Emit(OpCodes.Call,
typeof(AppDomain).GetProperty("FriendlyName").GetGetMethod());
      il.Emit(OpCodes.Call, typeof(Console).GetMethod("WriteLine",
                              new Type[] { typeof(String), typeof(String)
}));
      il.Emit(OpCodes.Ret);

      // Complete the HelloWorld type and save the assembly. The assembly
      // is placed in the location specified by DefineDynamicAssembly.
      Type myType = tb.CreateType();
      ab.Save(moduleName);
   }
```

```
}

/* This example produces output similar to the following:

DynamicBase is set to 'C:\DynamicAssemblyDir\5e4a7545'.
Dynamic directory is 'C:\DynamicAssemblyDir\5e4a7545\Example'.
Creating the dynamic directory.
Hello from 'MyDomain'!
 */
```

Remarks

This method sets the DynamicBase property of the internal AppDomainSetup associated with this instance.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- DynamicBase

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetPrincipalPolicy(PrincipalPolicy) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies how principal and identity objects should be attached to a thread if the thread attempts to bind to a principal while executing in this application domain.

C#

```
public void SetPrincipalPolicy (System.Security.Principal.PrincipalPolicy policy);
```

Parameters

policy PrincipalPolicy

One of the PrincipalPolicy values that specifies the type of the principal object to attach to threads.

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following example shows the effect on threads of using the SetPrincipalPolicy method to change the principal policy of the application domain. It also shows the effect of using the SetThreadPrincipal method to change the principal that is available for attaching to threads in the application domain.

C#

```
using System;
using System.Security.Principal;
using System.Threading;
```

```
class ADPrincipal
{
    static void Main(string[] args)
    {
        // Create a new thread with a generic principal.
        Thread t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Set the principal policy to WindowsPrincipal.
        AppDomain currentDomain = AppDomain.CurrentDomain;
        currentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

        // The new thread will have a Windows principal representing the
        // current user.
        t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Create a principal to use for new threads.
        IIdentity identity = new GenericIdentity("NewUser");
        IPrincipal principal = new GenericPrincipal(identity, null);
        currentDomain.SetThreadPrincipal(principal);

        // Create a new thread with the principal created above.
        t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Wait for user input before terminating.
        Console.ReadLine();
    }

    static void PrintPrincipalInformation()
    {
        IPrincipal curPrincipal = Thread.CurrentPrincipal;
        if(curPrincipal != null)
        {
            Console.WriteLine("Type: " + curPrincipal.GetType().Name);
            Console.WriteLine("Name: " + curPrincipal.Identity.Name);
            Console.WriteLine("Authenticated: " +
                curPrincipal.Identity.IsAuthenticated);
            Console.WriteLine();
        }
    }
}
```

Remarks

Setting this value will only be effective if you set it before using the Thread.CurrentPrincipal property. For example, if you set Thread.CurrentPrincipal to a

given principal (for example, a generic principal) and then use the SetPrincipalPolicy method to set the PrincipalPolicy to WindowsPrincipal, the current principal will remain the generic principal.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetShadowCopyFiles Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.SetShadowCopyFiles has been deprecated and is not supported.

Turns on shadow copying.

C#

```
[System.Obsolete("AppDomain.SetShadowCopyFiles has been deprecated and is not supported.")]
public void SetShadowCopyFiles ();
```

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

This method is now obsolete, and should not be used for new development.

C#

```
using System;
using System.Security.Policy;
namespace AppDomainSnippets
{
    class ADProperties
```

```
    {
        static void Main(string[] args)
        {

            AppDomainSetup setup = new AppDomainSetup();
            // Shadow copying will not work unless the application has a
name.
            setup.ApplicationName = "MyApplication";

            //Create evidence for the new application domain from evidence
of
            // current application domain.
            Evidence adevidence = AppDomain.CurrentDomain.Evidence;

            // Create a new application domain.
            AppDomain domain = AppDomain.CreateDomain("MyDomain",
adevidence, setup);

            // MyAssembly.dll is located in the Assemblies subdirectory.
            domain.AppendPrivatePath("Assemblies");
            // MyOtherAssembly.dll and MyThirdAssembly.dll are located in
the
            // MoreAssemblies subdirectory.
            domain.AppendPrivatePath("MoreAssemblies");
            // Display the relative search path.
            Console.WriteLine("RelativeSearchPath: " +
domain.RelativeSearchPath);
            // Because Load returns an Assembly object, the assemblies must
be
            // loaded into the current domain as well. This will fail unless
the
            // current domain also has these directories in its search path.
            AppDomain.CurrentDomain.AppendPrivatePath("Assemblies");
            AppDomain.CurrentDomain.AppendPrivatePath("MoreAssemblies");

            // Save shadow copies to C:\Cache
            domain.SetCachePath("C:\\Cache");
            // Shadow copy only the assemblies in the Assemblies directory.
            domain.SetShadowCopyPath(domain.BaseDirectory + "Assemblies");
            // Turn shadow copying on.
            domain.SetShadowCopyFiles();
            Console.WriteLine("ShadowCopyFiles turned on: " +
domain.ShadowCopyFiles);

            // This will be copied.
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly1 text name, Version, Culture,
PublicKeyToken");
            // This will not be copied.
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly2 text name, Version, Culture,
PublicKeyToken");

            // When the shadow copy path is cleared, the CLR will make
shadow copies
```

```
            // of all private assemblies.
            domain.ClearShadowCopyPath();
            // MoreAssemblies\MyThirdAssembly.dll should be shadow copied
this time.
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly3 text name, Version, Culture,
PublicKeyToken");

            // Unload the domain.
            AppDomain.Unload(domain);
        }
    }
}
```

Remarks

For more information on shadow copying, see Shadow Copying Assemblies.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- Shadow Copying Assemblies

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetShadowCopyPath(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> AppDomain.SetShadowCopyPath has been deprecated and is not supported.

Establishes the specified directory path as the location of assemblies to be shadow copied.

C#

```
[System.Obsolete("AppDomain.SetShadowCopyPath has been deprecated and is not supported.")]
public void SetShadowCopyPath (string? path);
```

Parameters

path String

A list of directory names, where each name is separated by a semicolon.

Attributes ObsoleteAttribute

Exceptions

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

This method is now obsolete, and should not be used for new development.

C#

```
using System;
using System.Security.Policy;
namespace AppDomainSnippets
{
    class ADShadowCopy
    {
        static void Main(string[] args)
        {

            AppDomainSetup setup = new AppDomainSetup();
            // Shadow copying will not work unless the application has a
name.
            setup.ApplicationName = "MyApplication";

            //Create evidence for the new application domain from evidence
of
            // current application domain.
            Evidence adevidence = AppDomain.CurrentDomain.Evidence;

            // Create a new application domain.
            AppDomain domain = AppDomain.CreateDomain("MyDomain",
adevidence, setup);

            // MyAssembly.dll is located in the Assemblies subdirectory.
            domain.AppendPrivatePath("Assemblies");
            // MyOtherAssembly.dll and MyThirdAssembly.dll are located in
the
            // MoreAssemblies subdirectory.
            domain.AppendPrivatePath("MoreAssemblies");
            // Display the relative search path.
            Console.WriteLine("RelativeSearchPath: " +
domain.RelativeSearchPath);
            // Because Load returns an Assembly object, the assemblies must
be
            // loaded into the current domain as well. This will fail unless
the
            // current domain also has these directories in its search path.
            AppDomain.CurrentDomain.AppendPrivatePath("Assemblies");
            AppDomain.CurrentDomain.AppendPrivatePath("MoreAssemblies");

            // Save shadow copies to C:\Cache
            domain.SetCachePath("C:\\Cache");
            // Shadow copy only the assemblies in the Assemblies directory.
            domain.SetShadowCopyPath(domain.BaseDirectory + "Assemblies");
            // Turn shadow copying on.
            domain.SetShadowCopyFiles();

            // This will be copied.
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly1 text name, Version, Culture,
PublicKeyToken");
            // This will not be copied.
```

```
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly2 text name, Version, Culture,
PublicKeyToken");

            // When the shadow copy path is cleared, the CLR will make
shadow copies
            // of all private assemblies.
            domain.ClearShadowCopyPath();
            // MoreAssemblies\MyThirdAssembly.dll should be shadow copied
this time.
            // You must supply a valid fully qualified assembly name here.
            domain.Load("Assembly3 text name, Version, Culture,
PublicKeyToken");

            // Unload the domain.
            AppDomain.Unload(domain);
        }
    }
}
```

Remarks

By default, a shadow copy includes all assemblies found through probing. The SetShadowCopyPath method restricts the shadow copy to the assemblies in the directories specified by `path`.

The SetShadowCopyPath method does not specify additional directories to be searched for assemblies. Assemblies to be shadow-copied must already be located in the search path, for example under BaseDirectory. The SetShadowCopyPath method specifies which search paths are eligible to be shadow-copied.

This method sets the ShadowCopyDirectories property of the internal AppDomainSetup associated with this instance.

For more information on shadow copying, see Shadow Copying Assemblies.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- Shadow Copying Assemblies

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.SetThread Principal(IPrincipal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the default principal object to be attached to threads if they attempt to bind to a principal while executing in this application domain.

C#

```
public void SetThreadPrincipal (System.Security.Principal.IPrincipal
principal);
```

Parameters

principal IPrincipal

The principal object to attach to threads.

Exceptions

ArgumentNullException

`principal` is `null`.

PolicyException

The thread principal has already been set.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

Examples

The following example shows the effect of using the SetThreadPrincipal method to change the principal that is available for attaching to threads that are executing in the application domain. It also shows the effect on threads of using the SetPrincipalPolicy method to change the principal policy of the application domain.

C#

```
using System;
using System.Security.Principal;
using System.Threading;

class ADPrincipal
{
    static void Main(string[] args)
    {
        // Create a new thread with a generic principal.
        Thread t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Set the principal policy to WindowsPrincipal.
        AppDomain currentDomain = AppDomain.CurrentDomain;
        currentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

        // The new thread will have a Windows principal representing the
        // current user.
        t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Create a principal to use for new threads.
        IIdentity identity = new GenericIdentity("NewUser");
        IPrincipal principal = new GenericPrincipal(identity, null);
        currentDomain.SetThreadPrincipal(principal);

        // Create a new thread with the principal created above.
        t = new Thread(new ThreadStart(PrintPrincipalInformation));
        t.Start();
        t.Join();

        // Wait for user input before terminating.
        Console.ReadLine();
    }

    static void PrintPrincipalInformation()
    {
        IPrincipal curPrincipal = Thread.CurrentPrincipal;
        if(curPrincipal != null)
        {
            Console.WriteLine("Type: " + curPrincipal.GetType().Name);
            Console.WriteLine("Name: " + curPrincipal.Identity.Name);
            Console.WriteLine("Authenticated: " +
                curPrincipal.Identity.IsAuthenticated);
            Console.WriteLine();
        }
    }
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Obtains a string representation that includes the friendly name of the application domain and any context policies.

C#

```
public override string ToString ();
```

Returns

String

A string formed by concatenating the literal string "Name:", the friendly name of the application domain, and either string representations of the context policies or the string "There are no context policies."

Exceptions

AppDomainUnloadedException

The application domain represented by the current AppDomain has been unloaded.

Examples

The following code example displays the return value of the ToString method.

C#

```
using System;
using System.Reflection;
using System.Security.Policy;
class ADSetup
{
    public static void Main()
    {
        // Create application domain setup information
```

```
        AppDomainSetup domaininfo = new AppDomainSetup();

        //Create evidence for the new appdomain from evidence of the current
application domain
        Evidence adevidence = AppDomain.CurrentDomain.Evidence;

        // Create appdomain
        AppDomain domain = AppDomain.CreateDomain("MyDomain", adevidence,
domaininfo);

        // Write out application domain information
        Console.WriteLine("Host domain: " +
AppDomain.CurrentDomain.FriendlyName);
        Console.WriteLine("child domain: " + domain.FriendlyName);
        Console.WriteLine("child domain name using ToString:" +
domain.ToString());
        Console.WriteLine();

        AppDomain.Unload(domain);
    }
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.Unload(AppDomain) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Creating and unloading AppDomains is not supported and throws an exception.

Unloads the specified application domain.

C#

```
[System.Obsolete("Creating and unloading AppDomains is not supported and throws an exception.", DiagnosticId="SYSLIB0024", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public static void Unload (AppDomain domain);
```

Parameters

`domain` AppDomain

An application domain to unload.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`domain` is `null`.

CannotUnloadAppDomainException

.NET Core and .NET 5+ only: In all cases.

-or-

`domain` could not be unloaded.

Exception

An error occurred during the unload process.

Examples

The following code example shows how to unload an application domain.

C#

```
using System;
using System.Reflection;
using System.Security.Policy;
class ADUnload
{
    public static void Main()
    {

        //Create evidence for the new appdomain.
        Evidence adevidence = AppDomain.CurrentDomain.Evidence;

        // Create the new application domain.
        AppDomain domain = AppDomain.CreateDomain("MyDomain", adevidence);

                Console.WriteLine("Host domain: " +
AppDomain.CurrentDomain.FriendlyName);
                Console.WriteLine("child domain: " + domain.FriendlyName);
        // Unload the application domain.
        AppDomain.Unload(domain);

        try
        {
        Console.WriteLine();
        // Note that the following statement creates an exception because
the domain no longer exists.
                Console.WriteLine("child domain: " + domain.FriendlyName);
        }

        catch (AppDomainUnloadedException e)
        {
        Console.WriteLine("The appdomain MyDomain does not exist.");
        }
    }
}
```

Remarks

In .NET Framework version 2.0 and later, there is a thread dedicated to unloading application domains. This improves reliability, especially when .NET Framework is hosted. When a thread calls Unload, the target domain is marked for unloading. The dedicated thread attempts to unload the domain, and all threads in the domain are aborted. If a thread does not abort, for example because it is executing unmanaged code, or because it is executing a `finally` block, then after a period of time, a CannotUnloadAppDomainException is thrown in the thread that originally called Unload. If the thread that could not be aborted eventually ends, the target domain is not unloaded. Thus, in .NET Framework version 2.0 and later, `domain` is not guaranteed to unload, because it might not be possible to terminate executing threads.

> **Note**
>
> In some cases, calling **Unload** causes an immediate **CannotUnloadAppDomainException**, for example if it is called in a finalizer.

The threads in `domain` are terminated using the Abort method, which throws a ThreadAbortException in the thread. Although the thread should terminate promptly, it can continue executing for an unpredictable amount of time in a `finally` clause.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Unavailable technologies on .NET Core/5+: Application domains

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

AppDomain.AssemblyLoad Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when an assembly is loaded.

C#

```
public event AssemblyLoadEventHandler? AssemblyLoad;
```

Event Type

AssemblyLoadEventHandler

Examples

The following sample demonstrates the AssemblyLoad event.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;

class AssemblyLoadSnippet {

   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      currentDomain.AssemblyLoad += new
AssemblyLoadEventHandler(MyAssemblyLoadEventHandler);

      PrintLoadedAssemblies(currentDomain);
      // Lists mscorlib and this assembly

      // You must supply a valid fully qualified assembly name here.
      currentDomain.CreateInstance("System.Windows.Forms, Version, Culture,
PublicKeyToken", "System.Windows.Forms.TextBox");
```

```
        // Loads System, System.Drawing, System.Windows.Forms

        PrintLoadedAssemblies(currentDomain);
        // Lists all five assemblies
    }

    static void PrintLoadedAssemblies(AppDomain domain) {
        Console.WriteLine("LOADED ASSEMBLIES:");
        foreach (Assembly a in domain.GetAssemblies()) {
            Console.WriteLine(a.FullName);
        }
        Console.WriteLine();
    }

    static void MyAssemblyLoadEventHandler(object sender,
AssemblyLoadEventArgs args) {
        Console.WriteLine("ASSEMBLY LOADED: " + args.LoadedAssembly.FullName);
        Console.WriteLine();
    }
}
```

Remarks

The AssemblyLoadEventHandler delegate for this event indicates what assembly was loaded.

To register an event handler for this event, you must have the required permissions, or a SecurityException is thrown.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

AppDomain.AssemblyResolve Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when the resolution of an assembly fails.

C#

```
public event ResolveEventHandler? AssemblyResolve;
```

Event Type

ResolveEventHandler

Examples

The following sample demonstrates the AssemblyResolve event.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
public class MyType
{
    public MyType()
    {
        Console.WriteLine();
        Console.WriteLine("MyType instantiated!");
    }
}

class AssemblyResolveSnippet
{
    public static void Main()
    {
        AppDomain currentDomain = AppDomain.CurrentDomain;

        // This call will fail to create an instance of MyType since the
```

```
        // assembly resolver is not set
        InstantiateMyTypeFail(currentDomain);

        currentDomain.AssemblyResolve += new
ResolveEventHandler(MyResolveEventHandler);

        // This call will succeed in creating an instance of MyType since
the
        // assembly resolver is now set.
        InstantiateMyTypeFail(currentDomain);

        // This call will succeed in creating an instance of MyType since
the
        // assembly name is valid.
        InstantiateMyTypeSucceed(currentDomain);
    }

    private static void InstantiateMyTypeFail(AppDomain domain)
    {
        // Calling InstantiateMyType will always fail since the assembly
info
        // given to CreateInstance is invalid.
        try
        {
            // You must supply a valid fully qualified assembly name here.
            domain.CreateInstance("Assembly text name, Version, Culture,
PublicKeyToken", "MyType");
        }
        catch (Exception e)
        {
            Console.WriteLine();
            Console.WriteLine(e.Message);
        }
    }

    private static void InstantiateMyTypeSucceed(AppDomain domain)
    {
        try
        {
            string asmname = Assembly.GetCallingAssembly().FullName;
            domain.CreateInstance(asmname, "MyType");
        }
        catch (Exception e)
        {
            Console.WriteLine();
            Console.WriteLine(e.Message);
        }
    }

    private static Assembly MyResolveEventHandler(object sender,
ResolveEventArgs args)
    {
        Console.WriteLine("Resolving...");
        return typeof(MyType).Assembly;
```

```
    }
}
```

Remarks

It is the responsibility of the ResolveEventHandler for this event to return the assembly that is specified by the ResolveEventArgs.Name property, or to return null if the assembly is not recognized. The assembly must be loaded into an execution context; if it is loaded into the reflection-only context, the load that caused this event to be raised fails.

For guidance on the use of this event, see Resolving Assembly Loads.

Beginning with the .NET Framework 4, the ResolveEventArgs.RequestingAssembly property returns the assembly that requested the assembly load that could not be resolved. For example, the loader might be unable to load a dependency of the requesting assembly because the requesting assembly and its dependency are not in the probing path. Knowing the identity of the requesting assembly might be useful in locating the dependency or in identifying the correct version, if more than one version of the dependency is available. For more information, see ResolveEventArgs.RequestingAssembly.

> **ⓘ Important**
>
> Beginning with the .NET Framework 4, the **ResolveEventHandler** event is raised for all assemblies, including resource assemblies. In earlier versions, the event was not raised for resource assemblies. If the operating system is localized, the handler might be called multiple times: once for each culture in the fallback chain.

For this event, the ResolveEventArgs.Name property returns the assembly name before policy is applied.

> **ⓘ Important**
>
> If more than one event handler is registered for this event, the event handlers are called in order until an event handler returns a value that isn't `null`. Subsequent event handlers are ignored.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RequestingAssembly
- Resolve assembly loads

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.DomainUnload Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when an AppDomain is about to be unloaded.

```
C#
public event EventHandler? DomainUnload;
```

Event Type

EventHandler

Remarks

The EventHandler delegate for this event can perform any termination activities before the application domain is unloaded.

Each application domain that needs to perform processing when it is unloaded should register an event handler for this event. A shared event handler should not be used, because the EventHandler delegate does not identify the domain that is being unloaded.

> **Note**
>
> This event is never raised in the default application domain.

Do not make assumptions about the thread the event is raised on. The event can be raised on a different thread than the one that called the Unload method.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.FirstChanceException Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when an exception is thrown in managed code, before the runtime searches the call stack for an exception handler in the application domain.

C#

```
public event
EventHandler<System.Runtime.ExceptionServices.FirstChanceExceptionEventArgs>
? FirstChanceException;
```

Event Type

EventHandler<FirstChanceExceptionEventArgs>

Examples

The following example creates a series of application domains named `AD0` through `AD3`, with a `Worker` object in each application domain. Each `Worker` object has a reference to the `Worker` object in the next application domain, except for the `Worker` in the last application domain. The FirstChanceException event is handled in all application domains except `AD1`.

> **Note**
>
> In addition to this example, which demonstrates first-chance exception notifications in multiple application domains, you can find simple use cases in **How to: Receive First-Chance Exception Notifications**.

When the application domains have been created, the default application domain calls the `TestException` method for the first application domain. Each `Worker` object calls the `TestException` method for the next application domain, until the last `Worker` throws an exception that is either handled or unhandled. Thus, the current thread passes through

all the application domains, and `TestException` is added to the stack in each application domain.

When the last `Worker` object handles the exception, the FirstChanceException event is raised only in the last application domain. The other application domains never get a chance to handle the exception, so the event is not raised.

When the last `Worker` object does not handle the exception, the FirstChanceException event is raised in each application domain that has an event handler. After each event handler has finished, the stack continues to unwind until the exception is caught by the default application domain.

> **Note**
>
> To see how the stack display grows as the event is raised closer and closer to the default application domain, change `e.Exception.Message` to `e.Exception` in the `FirstChanceHandler` event handlers. Notice that when `TestException` is called across application domain boundaries, it appears twice: once for the proxy and once for the stub.

C#

```
using System;
using System.Reflection;
using System.Runtime.ExceptionServices;

class FirstChanceExceptionSnippet
{
    static void Main()
    {
        AppDomain.CurrentDomain.FirstChanceException += FirstChanceHandler;

        // Create a set of application domains, with a Worker object in each
one.
        // Each Worker object creates the next application domain.
        AppDomain ad = AppDomain.CreateDomain("AD0");
        Worker w = (Worker) ad.CreateInstanceAndUnwrap(
                                typeof(Worker).Assembly.FullName, "Worker");
        w.Initialize(0, 3);

        Console.WriteLine("\r\nThe last application domain throws an
exception and catches it:");
        Console.WriteLine();
        w.TestException(true);

        try
        {
```

```
            Console.WriteLine(
                "\r\nThe last application domain throws an exception and
does not catch it:");
            Console.WriteLine();
            w.TestException(false);
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine("ArgumentException caught in {0}: {1}",
                AppDomain.CurrentDomain.FriendlyName, ex.Message);
        }
    }

    static void FirstChanceHandler(object source,
FirstChanceExceptionEventArgs e)
    {
        Console.WriteLine("FirstChanceException event raised in {0}: {1}",
            AppDomain.CurrentDomain.FriendlyName, e.Exception.Message);
    }
}

public class Worker : MarshalByRefObject
{
    private AppDomain ad = null;
    private Worker w = null;

    public void Initialize(int count, int max)
    {
        // Handle the FirstChanceException event in all application domains
except
        // AD1.
        if (count != 1)
        {
            AppDomain.CurrentDomain.FirstChanceException +=
FirstChanceHandler;
        }

        // Create another application domain, until the maximum is reached.
        // Field w remains null in the last application domain, as a signal
        // to TestException().
        if (count < max)
        {
            int next = count + 1;
            ad = AppDomain.CreateDomain("AD" + next);
            w = (Worker) ad.CreateInstanceAndUnwrap(
                                typeof(Worker).Assembly.FullName, "Worker");
            w.Initialize(next, max);
        }
    }

    public void TestException(bool handled)
    {
        // As long as there is another application domain, call
TestException() on
        // its Worker object. When the last application domain is reached,
```

```
throw a
        // handled or unhandled exception.
        if (w != null)
        {
            w.TestException(handled);
        }
        else if (handled)
        {
            try
            {
                throw new ArgumentException("Thrown in " +
AppDomain.CurrentDomain.FriendlyName);
            }
            catch (ArgumentException ex)
            {
                Console.WriteLine("ArgumentException caught in {0}: {1}",
                    AppDomain.CurrentDomain.FriendlyName, ex.Message);
            }
        }
        else
        {
            throw new ArgumentException("Thrown in " +
AppDomain.CurrentDomain.FriendlyName);
        }
    }

    static void FirstChanceHandler(object source,
FirstChanceExceptionEventArgs e)
    {
        Console.WriteLine("FirstChanceException event raised in {0}: {1}",
            AppDomain.CurrentDomain.FriendlyName, e.Exception.Message);
    }
}

/* This example produces output similar to the following:

The last application domain throws an exception and catches it:

FirstChanceException event raised in AD3: Thrown in AD3
ArgumentException caught in AD3: Thrown in AD3

The last application domain throws an exception and does not catch it:

FirstChanceException event raised in AD3: Thrown in AD3
FirstChanceException event raised in AD2: Thrown in AD3
FirstChanceException event raised in AD0: Thrown in AD3
FirstChanceException event raised in Example.exe: Thrown in AD3
ArgumentException caught in Example.exe: Thrown in AD3
 */
```

Remarks

This event is only a notification. Handling this event does not handle the exception or affect subsequent exception handling in any way. After the event has been raised and event handlers have been invoked, the common language runtime (CLR) begins to search for a handler for the exception. FirstChanceException provides the application domain with a first chance to examine any managed exception.

The event can be handled per application domain. If a thread passes through multiple application domains while executing a call, the event is raised in each application domain that has registered an event handler, before the CLR begins searching for a matching exception handler in that application domain. After the event has been handled, a search is made for a matching exception handler in that application domain. If none is found, the event is raised in the next application domain.

You must handle all exceptions that occur in the event handler for the FirstChanceException event. Otherwise, FirstChanceException is raised recursively. This could result in a stack overflow and termination of the application. We recommend that you implement event handlers for this event as constrained execution regions (CERs), to keep infrastructure-related exceptions such as out-of-memory or stack overflow from affecting the virtual machine while the exception notification is being processed.

This event is not raised for exceptions that indicate corruption of process state, such as access violations, unless the event handler is security-critical and has the HandleProcessCorruptedStateExceptionsAttribute attribute.

The common language runtime suspends thread aborts while this notification event is being handled.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FirstChanceExceptionEventArgs
- UnhandledException
- How to: Receive First-Chance Exception Notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomain.ProcessExit Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when the default application domain's parent process exits.

```
C#
public event EventHandler? ProcessExit;
```

Event Type

EventHandler

Remarks

The EventHandler for this event can perform termination activities, such as closing files, releasing storage and so on, before the process ends.

Beginning with the .NET Framework version 2.0, this event is raised in each application domain that registers an event handler.

> **Note**
>
> In .NET Framework, the total execution time of all **ProcessExit** event handlers is limited, just as the total execution time of all finalizers is limited at process shutdown. The default is two seconds. An unmanaged host can change this execution time by calling the **ICLRPolicyManager::SetTimeout** method with the **OPR_ProcessExit** enumeration value. This time limit does not exist in .NET Core.

In the .NET Framework versions 1.0 and 1.1, this event is raised only in the default application domain, and only if an event handler is registered in the default application domain.

To register an event handler for this event, you must have the required permissions, or a SecurityException is thrown.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ReflectionOnlyAssemblyResolve Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when the resolution of an assembly fails in the reflection-only context.

C#

```
public event ResolveEventHandler? ReflectionOnlyAssemblyResolve;
```

Event Type

ResolveEventHandler

Remarks

In the reflection-only context, dependencies are not resolved automatically. They must be preloaded or returned by the handler for this event. This event is raised when an assembly has a dependency that is not already loaded into the reflection-only context. The missing dependency is specified by the ResolveEventArgs.Name property. The ResolveEventHandler for this event must return an assembly that satisfies the dependency. The assembly that is returned must be loaded into the reflection-only context.

> **Important**
>
> This event is raised only for missing dependencies of the assembly that you are loading into the reflection-only context (for example, by using the **Assembly.ReflectionOnlyLoad** method). It is not raised if the assembly that you are loading cannot be found.

Beginning with the .NET Framework 4, the ResolveEventArgs.RequestingAssembly property returns the assembly that requested the assembly load that could not be

resolved. Knowing the identity of the requesting assembly might be useful in identifying the correct version of the dependency, if more than one version is available. For more information, see ResolveEventArgs.RequestingAssembly.

For this event, the ResolveEventArgs.Name property returns the assembly name before policy is applied.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RequestingAssembly

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.ResourceResolve Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when the resolution of a resource fails because the resource is not a valid linked or embedded resource in the assembly.

```
C#
public event ResolveEventHandler? ResourceResolve;
```

Event Type

ResolveEventHandler

Remarks

The ResolveEventHandler for this event can attempt to locate the assembly containing the resource and return it.

> **Important**
>
> This event is not raised if resolution fails because no file can be found for a valid linked resource. It is raised if a manifest resource stream cannot be found, but it is not raised if an individual resource key cannot be found.

Beginning with the .NET Framework 4, the ResolveEventArgs.RequestingAssembly property contains the assembly that requested the resource. For more information, see ResolveEventArgs.RequestingAssembly.

To register an event handler for this event, you must have the required permissions, or a SecurityException is thrown.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RequestingAssembly

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.TypeResolve Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when the resolution of a type fails.

C#

```
public event ResolveEventHandler? TypeResolve;
```

Event Type

ResolveEventHandler

Examples

The following sample demonstrates the TypeResolve event.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;
using System.Reflection.Emit;

class Test
{
    // For this code example, the following information needs to be
    // available to both Main and the HandleTypeResolve event
    // handler:
    private static AssemblyBuilder ab;
    private static string moduleName;

    public static void Main()
    {
        AppDomain currDom = AppDomain.CurrentDomain;
```

```
        // Create a dynamic assembly with one module, to be saved to
        // disk (AssemblyBuilderAccess.Save).
        //
        AssemblyName aName = new AssemblyName();
        aName.Name = "Transient";
        moduleName = aName.Name + ".dll";
        ab = currDom.DefineDynamicAssembly(aName,
            AssemblyBuilderAccess.Save);
        ModuleBuilder mb = ab.DefineDynamicModule(aName.Name, moduleName);

        // The dynamic assembly has just one dummy type, to demonstrate
        // type resolution.
        TypeBuilder tb = mb.DefineType("Example");
        tb.CreateType();

        // First, try to load the type without saving the dynamic
        // assembly and without hooking up the TypeResolve event. The
        // type cannot be loaded.
        try
        {
            Type temp = Type.GetType("Example", true);
            Console.WriteLine("Loaded type {0}.", temp);
        }
        catch (TypeLoadException)
        {
            Console.WriteLine("Loader could not resolve the type.");
        }

        // Hook up the TypeResolve event.
        //
        currDom.TypeResolve +=
            new ResolveEventHandler(HandleTypeResolve);

        // Now try to load the type again. The TypeResolve event is
        // raised, the dynamic assembly is saved, and the dummy type is
        // loaded successfully. Display it to the console, and create
        // an instance.
        Type t = Type.GetType("Example", true);
        Console.WriteLine("Loaded type \"{0}\".", t);
        Object o = Activator.CreateInstance(t);
    }

    static Assembly HandleTypeResolve(object sender, ResolveEventArgs args)
    {
        Console.WriteLine("TypeResolve event handler.");

        // Save the dynamic assembly, and then load it using its
        // display name. Return the loaded assembly.
        //
        ab.Save(moduleName);
        return Assembly.Load(ab.FullName);
    }
}

/* This code example produces the following output:
```

```
Loader could not resolve the type.
TypeResolve event handler.
Loaded type "Example".
 */
```

Remarks

The TypeResolve event occurs when the common language runtime is unable to determine the assembly that can create the requested type. This can occur if the type is defined in a dynamic assembly, or the type is not defined in a dynamic assembly but the runtime does not know which assembly the type is defined in. The latter situation can occur when Type.GetType is called with a type name that is not qualified with the assembly name.

The ResolveEventHandler for this event can attempt to locate and create the type.

However, the TypeResolve event does not occur if the runtime knows it is not possible to find a type in certain assemblies. For example, this event does not occur if the type is not found in a static assembly because the runtime knows types cannot be added dynamically to static assemblies.

Beginning with the .NET Framework 4, the ResolveEventArgs.RequestingAssembly property contains the assembly that requested the type. For more information, see ResolveEventArgs.RequestingAssembly.

To register an event handler for this event, you must have the required permissions, or a SecurityException is thrown.

For more information about handling events, see Handling and Raising Events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RequestingAssembly

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomain.UnhandledException Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Occurs when an exception is not caught.

C#

```
public event UnhandledExceptionEventHandler? UnhandledException;
```

Event Type

UnhandledExceptionEventHandler

Examples

The following example demonstrates the UnhandledException event. It defines an event handler, `MyHandler`, that is invoked whenever an unhandled exception is thrown in the default application domain. It then throws two exceptions. The first is handled by a **try/catch** block. The second is unhandled and invokes the `MyHandle` routine before the application terminates.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      currentDomain.UnhandledException += new
UnhandledExceptionEventHandler(MyHandler);

      try {
         throw new Exception("1");
      } catch (Exception e) {
         Console.WriteLine("Catch clause caught : {0} \n", e.Message);
      }
```

```
        throw new Exception("2");
    }

    static void MyHandler(object sender, UnhandledExceptionEventArgs args)
    {
        Exception e = (Exception) args.ExceptionObject;
        Console.WriteLine("MyHandler caught : " + e.Message);
        Console.WriteLine("Runtime terminating: {0}", args.IsTerminating);
    }
}
// The example displays the following output:
//       Catch clause caught : 1
//
//       MyHandler caught : 2
//       Runtime terminating: True
//
//       Unhandled Exception: System.Exception: 2
//          at Example.Main()
```

Remarks

For more information about this API, see Supplemental API remarks for UnhandledException.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomainSetup Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents assembly binding information that can be added to an instance of AppDomain.

C#

```
public sealed class AppDomainSetup
```

Inheritance Object → AppDomainSetup

Remarks

Changing the properties of an AppDomainSetup instance does not affect any existing AppDomain. It can affect only the creation of a new AppDomain, when the CreateDomain method is called with the AppDomainSetup instance as a parameter.

This class implements the IAppDomainSetup interface.

> ⊗ **Caution**
>
> The default value for the **DisallowCodeDownload** property is false. This setting is unsafe for services. To help prevent services from downloading partially trusted code, set this property to true

Properties

Expand table

ApplicationBase	Gets the name of the directory containing the application.
TargetFrameworkName	Gets (or, in .NET Framework, sets) a string that specifies the target framework in a format that can be parsed by the FrameworkName(String) constructor.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

See also

- How To: Configure an Application Domain

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomainSetup.ApplicationBase Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the directory containing the application.

C#

```
public string? ApplicationBase { get; }
```

Property Value

String

The name of the application base directory.

Examples

The following example demonstrates how to use the ApplicationBase property to set the location where the assembly loader begins probing for assemblies to load into a new application domain.

> **Note**
>
> You must ensure that the folder you specify exists.

C#

```
using System;

class ADSetupInformation
{
    static void Main()
    {
        AppDomain root = AppDomain.CurrentDomain;
```

```
        AppDomainSetup setup = new AppDomainSetup();
        setup.ApplicationBase =
            root.SetupInformation.ApplicationBase + @"MyAppSubfolder\";

        AppDomain domain = AppDomain.CreateDomain("MyDomain", null, setup);

        Console.WriteLine("Application base of {0}:\r\n\t{1}",
            root.FriendlyName, root.SetupInformation.ApplicationBase);
        Console.WriteLine("Application base of {0}:\r\n\t{1}",
            domain.FriendlyName, domain.SetupInformation.ApplicationBase);

        AppDomain.Unload(domain);
    }
}

/* This example produces output similar to the following:

Application base of MyApp.exe:
        C:\Program Files\MyApp\
Application base of MyDomain:
        C:\Program Files\MyApp\MyAppSubfolder\
 */
```

Remarks

The application base directory is where the assembly manager begins probing for assemblies.

The ApplicationBase property can influence which permissions are granted to an application domain. For example, an application domain originating from the local computer normally receives full trust based on its location of origin. However, if the ApplicationBase property of that AppDomain is set to the full name of an intranet directory, the ApplicationBase setting restricts the permissions granted to the application domain to a LocalIntranet grant even though the application domain actually originates from the local computer.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AppDomainSetup.TargetFramework Name Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets (or, in .NET Framework, sets) a string that specifies the target framework in a format that can be parsed by the FrameworkName(String) constructor.

```
C#

public string? TargetFrameworkName { get; }
```

Property Value

String

The target framework of the executable that started the process.

Remarks

Set this property to tell the currently running version of the .NET Framework that the application domain will be used to load and run assemblies that have been compiled for the target version and profile. If the currently running version of the .NET Framework has optional behaviors that affect the compatibility of code that was compiled with the target version, it can enable or disable those behaviors, as appropriate, to improve compatibility. This can be useful when an application supports add-ins compiled with multiple versions of the .NET Framework by running them in separate application domains.

For client applications, the value of the TargetFrameworkName property is inferred from the TargetFrameworkAttribute attribute. In Visual Studio 2010 and later, this attribute is added to the assembly automatically based on the setting of the project's **Target Framework** property.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomainUnloadedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt is made to access an unloaded application domain.

```
C#
public class AppDomainUnloadedException : SystemException
```

Inheritance Object → Exception → SystemException → AppDomainUnloadedException

Examples

This section contains two code examples. The first example demonstrates the effects of an AppDomainUnloadedException on various threads, and the second shows elementary application domain unloading.

Example 1

The following code example defines a `TestClass` class that can be marshaled across application domain boundaries and an `Example` class containing a `static` (`Shared` in Visual Basic) `ThreadProc` method. The `ThreadProc` method creates an application domain, creates a `TestClass` object in the domain, and calls a method of `TestClass` that unloads the executing domain, causing an AppDomainUnloadedException.

The `TestClass` method is executed without exception handling from a ThreadPool thread and from an ordinary thread, demonstrating that the unhandled exception terminates the task or thread but not the application. It is then executed with and without exception handling from the main application thread, demonstrating that it terminates the application if not handled.

```
C#
```

```
using System;
using System.Threading;
using System.Runtime.InteropServices;

public class Example
{
    public static void Main()
    {
        // 1. Queue ThreadProc as a task for a ThreadPool thread.
        ThreadPool.QueueUserWorkItem(ThreadProc, " from a ThreadPool
thread");
        Thread.Sleep(1000);

        // 2. Execute ThreadProc on an ordinary thread.
        Thread t = new Thread(ThreadProc);
        t.Start(" from an ordinary thread");
        t.Join();

        // 3. Execute ThreadProc on the main thread, with
        //    exception handling.
        try
        {
            ThreadProc(" from the main application thread (handled)");
        }
        catch (AppDomainUnloadedException adue)
        {
            Console.WriteLine("Main thread caught
AppDomainUnloadedException: {0}", adue.Message);
        }

        // 4. Execute ThreadProc on the main thread without
        //    exception handling.
        ThreadProc(" from the main application thread (unhandled)");

        Console.WriteLine("Main: This message is never displayed.");
    }

    private static void ThreadProc(object state)
    {
        // Create an application domain, and create an instance
        // of TestClass in the application domain. The first
        // parameter of CreateInstanceAndUnwrap is the name of
        // this executable. If you compile the example code using
        // any name other than "Sample.exe", you must change the
        // parameter appropriately.
        AppDomain ad = AppDomain.CreateDomain("TestDomain");
        TestClass tc = (TestClass)ad.CreateInstanceAndUnwrap("Sample",
"TestClass");

        // In the new application domain, execute a method that
        // unloads the AppDomain. The unhandled exception this
        // causes ends the current thread.
        tc.UnloadCurrentDomain(state);
```

```
        Console.WriteLine("ThreadProc: This message is never displayed.");
    }
}

// TestClass derives from MarshalByRefObject, so it can be marshaled
// across application domain boundaries.
//
public class TestClass : MarshalByRefObject
{
    public void UnloadCurrentDomain(object state)
    {
        Console.WriteLine("\nUnloading the current AppDomain{0}.", state);

        // Unload the current application domain. This causes
        // an AppDomainUnloadedException to be thrown.
        //
        AppDomain.Unload(AppDomain.CurrentDomain);
    }
}

/* This code example produces output similar to the following:
Unloading the current AppDomain from a ThreadPool thread.

Unloading the current AppDomain from an ordinary thread.

Unloading the current AppDomain from the main application thread (handled).
Main thread caught AppDomainUnloadedException: The application domain in
which the thread was running has been unloaded.

Unloading the current AppDomain from the main application thread
(unhandled).

Unhandled Exception: System.AppDomainUnloadedException: The application
domain in which the thread was running has been unloaded.
   at TestClass.UnloadCurrentDomain(Object state)
   at Example.ThreadProc(Object state)
   at Example.Main()
 */
```

Example 2

The following code example creates and unloads an application domain, and demonstrates that an AppDomainUnloadedException is thrown on a subsequent attempt to access the unloaded domain.

C#

```
using System;
using System.Reflection;
using System.Security.Policy;
class ADUnload
{
```

```
    public static void Main()
    {

        //Create evidence for the new appdomain.
        Evidence adevidence = AppDomain.CurrentDomain.Evidence;

        // Create the new application domain.
        AppDomain domain = AppDomain.CreateDomain("MyDomain", adevidence);

                Console.WriteLine("Host domain: " +
AppDomain.CurrentDomain.FriendlyName);
                Console.WriteLine("child domain: " + domain.FriendlyName);
        // Unload the application domain.
        AppDomain.Unload(domain);

        try
        {
        Console.WriteLine();
        // Note that the following statement creates an exception because
the domain no longer exists.
                Console.WriteLine("child domain: " + domain.FriendlyName);
        }

        catch (AppDomainUnloadedException e)
        {
        Console.WriteLine("The appdomain MyDomain does not exist.");
        }
    }
}
```

Remarks

In the .NET Framework version 2.0, an AppDomainUnloadedException that is not handled in user code has the following effect:

- If a thread was started in managed code, it is terminated. The unhandled exception is not allowed to terminate the application.
- If a task is executing on a ThreadPool thread, it is terminated and the thread is returned to the thread pool. The unhandled exception is not allowed to terminate the application.
- If a thread started in unmanaged code, such as the main application thread, it is terminated. The unhandled exception is allowed to proceed, and the operating system terminates the application.

AppDomainUnloadedException uses the HRESULT COR_E_APPDOMAINUNLOADED, which has the value 0x80131014.

For a list of initial property values for an instance of AppDomainUnloadedException, see the AppDomainUnloadedException constructors.

Constructors

Expand table

AppDomainUnloadedException()	Initializes a new instance of the AppDomainUnloadedException class.
AppDomainUnloadedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AppDomainUnloadedException class with serialized data.
AppDomainUnloadedException(String)	Initializes a new instance of the AppDomainUnloadedException class with a specified error message.
AppDomainUnloadedException(String, Exception)	Initializes a new instance of the AppDomainUnloadedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error.

	(Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception.

(Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- AppDomain
- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AppDomainUnloadedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the AppDomainUnloadedException class.

Overloads

Expand table

AppDomainUnloadedException()	Initializes a new instance of the AppDomainUnloadedException class.
AppDomainUnloadedException(String)	Initializes a new instance of the AppDomainUnloadedException class with a specified error message.
AppDomainUnloadedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the AppDomainUnloadedException class with serialized data.
AppDomainUnloadedException(String, Exception)	Initializes a new instance of the AppDomainUnloadedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

AppDomainUnloadedException()

Initializes a new instance of the AppDomainUnloadedException class.

C#

```
public AppDomainUnloadedException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The target application domain has been unloaded." This message takes into account the current system culture.

The following table shows the initial property values for an instance of AppDomainUnloadedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AppDomainUnloadedException(String)

Initializes a new instance of the AppDomainUnloadedException class with a specified error message.

C#

```
public AppDomainUnloadedException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of AppDomainUnloadedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AppDomainUnloadedException(SerializationInfo, StreamingContext)

> **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the AppDomainUnloadedException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected AppDomainUnloadedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

AppDomainUnloadedException(String, Exception)

Initializes a new instance of the AppDomainUnloadedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public AppDomainUnloadedException (string? message, Exception?
innerException);
```

Parameters

`message` String

The message that describes the error.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of AppDomainUnloadedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as the base class for application-defined exceptions.

C#

```
public class ApplicationException : Exception
```

Inheritance Object → Exception → ApplicationException

Derived Microsoft.JScript.BreakOutOfFinally
Microsoft.JScript.ContinueOutOfFinally
Microsoft.JScript.JScriptException
Microsoft.JScript.NoContextException
Microsoft.JScript.ReturnOutOfFinally
More...

Remarks

Important

You should derive custom exceptions from the **Exception** class rather than the **ApplicationException** class. You should not throw an **ApplicationException** exception in your code, and you should not catch an **ApplicationException** exception unless you intend to re-throw the original exception.

Constructors

Expand table

ApplicationException()	Initializes a new instance of the ApplicationException class.

ApplicationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ApplicationException class with serialized data.
ApplicationException(String)	Initializes a new instance of the ApplicationException class with a specified error message.
ApplicationException(String, Exception)	Initializes a new instance of the ApplicationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- Handling and Throwing Exceptions
- Using Standard Exception Types

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ApplicationException class.

Overloads

Expand table

ApplicationException()	Initializes a new instance of the ApplicationException class.
ApplicationException(String)	Initializes a new instance of the ApplicationException class with a specified error message.
ApplicationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ApplicationException class with serialized data.
ApplicationException(String, Exception)	Initializes a new instance of the ApplicationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

ApplicationException()

Initializes a new instance of the ApplicationException class.

C#

```
public ApplicationException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "An application error has occurred." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ApplicationException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ApplicationException(String)

Initializes a new instance of the ApplicationException class with a specified error message.

C#

```
public ApplicationException (string? message);
```

Parameters

`message` String

A message that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This message takes into account the current system culture.

The following table shows the initial property values for an instance of ApplicationException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
`Message`	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ApplicationException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ApplicationException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ApplicationException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and Soap Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ApplicationException(String, Exception)

Initializes a new instance of the ApplicationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public ApplicationException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ApplicationException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Contains information used to uniquely identify a manifest-based application. This class cannot be inherited.

C#

```
public sealed class ApplicationId
```

Inheritance Object → ApplicationId

Examples

The following code example displays the ApplicationId properties by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application.

C#

```
using System;
using System.Collections;
using System.Text;
using System.Security.Policy;
using System.Reflection;
using System.Security;

namespace ActivationContextSample
{
    public class Program : MarshalByRefObject
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Full name = " +

AppDomain.CurrentDomain.ActivationContext.Identity.FullName);
            Console.WriteLine("Code base = " +

AppDomain.CurrentDomain.ActivationContext.Identity.CodeBase);
```

```
            ApplicationSecurityInfo asi = new
ApplicationSecurityInfo(AppDomain.CurrentDomain.ActivationContext);

            Console.WriteLine("ApplicationId.Name property = " +
asi.ApplicationId.Name);
            if (asi.ApplicationId.Culture != null)
                Console.WriteLine("ApplicationId.Culture property = " +
asi.ApplicationId.Culture.ToString());
            Console.WriteLine("ApplicationId.ProcessorArchitecture property
= " + asi.ApplicationId.ProcessorArchitecture);
            Console.WriteLine("ApplicationId.Version property = " +
asi.ApplicationId.Version);
            // To display the value of the public key, enumerate the Byte
array for the property.
            Console.Write("ApplicationId.PublicKeyToken property = ");
            byte[] pk = asi.ApplicationId.PublicKeyToken;
            for (int i = 0; i < pk.GetLength(0); i++)
                Console.Write("{0:x}", pk[i]);

            Console.Read();
        }

        public void Run()
        {
            Main(new string[] { });
            Console.ReadLine();
        }
    }
}
```

Remarks

An application identity consists of the publisher's public key, the application name, the target processor architecture, the application culture, and the application version. The application identity properties values correspond to information contained in the application manifest. For more information about the application manifest, see ClickOnce Application Manifest.

ApplicationId is used by the ApplicationSecurityInfo class to identify a manifest-based application.

Constructors

Expand table

ApplicationId(Byte[], String, Version, String, String)	Initializes a new instance of the ApplicationId class.

Properties

Expand table

Culture	Gets a string representing the culture information for the application.
Name	Gets the name of the application.
ProcessorArchitecture	Gets the target processor architecture for the application.
PublicKeyToken	Gets the public key token for the application.
Version	Gets the version of the application.

Methods

Expand table

Copy()	Creates and returns an identical copy of the current application identity.
Equals(Object)	Determines whether the specified ApplicationId object is equivalent to the current ApplicationId.
GetHashCode()	Gets the hash code for the current application identity.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the application identity.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ApplicationId(Byte[], String, Version, String, String) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ApplicationId class.

C#

```
public ApplicationId (byte[] publicKeyToken, string name, Version version,
string? processorArchitecture, string? culture);
```

Parameters

publicKeyToken Byte[]

The array of bytes representing the raw public key data.

name String

The name of the application.

version Version

A Version object that specifies the version of the application.

processorArchitecture String

The processor architecture of the application.

culture String

The culture of the application.

Exceptions

ArgumentNullException

`name` is `null`.

-or-

`version` is `null`.

-or-

`publicKeyToken` is `null`.

ArgumentException

`name` is an empty string.

Remarks

The constructor initializes the Culture, Name, ProcessorArchitecture, PublicKeyToken, and Version properties of the ApplicationId.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.Culture Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a string representing the culture information for the application.

```
C#

public string? Culture { get; }
```

Property Value

String

The culture information for the application.

Examples

The following code example displays the Culture property by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application. This code example is part of a larger example provided for the ApplicationId class.

```
C#

if (asi.ApplicationId.Culture != null)
    Console.WriteLine("ApplicationId.Culture property = " +
asi.ApplicationId.Culture.ToString());
```

Remarks

The Culture value can include language information, for example, "en-US".

For information about culture values, see CultureInfo.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.Name Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the application.

C#

```
public string Name { get; }
```

Property Value

String

The name of the application.

Examples

The following code example displays the Name property by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application. This code example is part of a larger example provided for the ApplicationId class.

C#

```
ApplicationSecurityInfo asi = new
ApplicationSecurityInfo(AppDomain.CurrentDomain.ActivationContext);

Console.WriteLine("ApplicationId.Name property = " +
asi.ApplicationId.Name);
```

Remarks

Name is the simple, unencrypted name of the application.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.ProcessorArchitecture Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the target processor architecture for the application.

C#

```
public string? ProcessorArchitecture { get; }
```

Property Value

String

The processor architecture of the application.

Examples

The following code example displays the ProcessorArchitecture property by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application. This code example is part of a larger example provided for the ApplicationId class.

C#

```
Console.WriteLine("ApplicationId.ProcessorArchitecture property = " +
asi.ApplicationId.ProcessorArchitecture);
```

Remarks

Valid values include "msil", "x86", "ia64", and "amd64". The default is "msil", or Microsoft intermediate language, which means all your assemblies are platform independent, and the common language runtime will compile them when they are needed by your application.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.PublicKeyToken Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the public key token for the application.

C#

```
public byte[] PublicKeyToken { get; }
```

Property Value

Byte[]

A byte array containing the public key token for the application.

Examples

The following code example displays the PublicKeyToken property by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application. This code example is part of a larger example provided for the ApplicationId class.

C#

```
// To display the value of the public key, enumerate the Byte array for the
property.
Console.Write("ApplicationId.PublicKeyToken property = ");
byte[] pk = asi.ApplicationId.PublicKeyToken;
for (int i = 0; i < pk.GetLength(0); i++)
    Console.Write("{0:x}", pk[i]);
```

Remarks

The public key token is represented by the last 8 bytes of the SHA-1 hash of the public key under which the application is signed.

For more information, see AssemblyName.GetPublicKeyToken.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.Version Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the version of the application.

```
C#
public Version Version { get; }
```

Property Value

Version

A Version that specifies the version of the application.

Examples

The following code example displays the Version property by obtaining the ApplicationId from an ApplicationSecurityInfo instance created using the ActivationContext for the currently executing manifest-based application. This code example is part of a larger example provided for the ApplicationId class.

```
C#
Console.WriteLine("ApplicationId.Version property = " +
asi.ApplicationId.Version);
```

Remarks

For more information, see System.Version.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ApplicationId.Copy Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates and returns an identical copy of the current application identity.

C#

```
public ApplicationId Copy ();
```

Returns

ApplicationId

An ApplicationId object that represents an exact copy of the original.

Remarks

The copy returned by this method is equal to the original as substantiated by the Equals method. The hash codes for the original and the copy are also equal.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ApplicationId.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified ApplicationId object is equivalent to the current ApplicationId.

```
C#
public override bool Equals (object? o);
```

Parameters

o Object

The ApplicationId object to compare to the current ApplicationId.

Returns

Boolean

`true` if the specified ApplicationId object is equivalent to the current ApplicationId; otherwise, `false`.

Remarks

This method overrides the Object.Equals(Object) method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ApplicationId.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hash code for the current application identity.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current application identity.

Remarks

This method generates the same hash code for two objects that have the same values for their Name and Version properties.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ApplicationId.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates and returns a string representation of the application identity.

C#

```
public override string ToString ();
```

Returns

String

A string representation of the application identity.

Remarks

This method overrides the Object.ToString method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ArgIterator Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a variable-length argument list; that is, the parameters of a function that takes a variable number of arguments.

```
C#
public ref struct ArgIterator
```

Inheritance Object → ValueType → ArgIterator

Remarks

Developers who write compilers use the ArgIterator structure to enumerate the mandatory and optional arguments in an argument list. The ArgIterator structure is not generally useful for applications other than compilers.

The functionality in the ArgIterator structure is typically hidden in the syntax of a specific programming language. For example, in the C++ programming language you declare a variable-length argument list by specifying an ellipsis ("...") at the end of the argument list. The ArgIterator structure is useful primarily when a development language does not provide direct support for accessing variable-length parameters.

Constructors

Expand table

ArgIterator(RuntimeArgumentHandle)	Initializes a new instance of the ArgIterator structure using the specified argument list.
ArgIterator(RuntimeArgumentHandle, Void*)	Initializes a new instance of the ArgIterator structure using the specified argument list and a pointer to an item in the list.

Methods

Expand table

End()	Concludes processing of the variable-length argument list represented by this instance.
Equals(Object)	This method is not supported, and always throws NotSupportedException.
GetHashCode()	Returns the hash code of this object.
GetNextArg()	Returns the next argument in a variable-length argument list.
GetNextArg(RuntimeTypeHandle)	Returns the next argument in a variable-length argument list that has a specified type.
GetNextArgType()	Returns the type of the next argument.
GetRemainingCount()	Returns the number of arguments remaining in the argument list.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgIterator Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArgIterator structure.

Overloads

Expand table

ArgIterator(RuntimeArgumentHandle)	Initializes a new instance of the ArgIterator structure using the specified argument list.
ArgIterator(RuntimeArgumentHandle, Void*)	Initializes a new instance of the ArgIterator structure using the specified argument list and a pointer to an item in the list.

ArgIterator(RuntimeArgumentHandle)

Initializes a new instance of the ArgIterator structure using the specified argument list.

C#

```
public ArgIterator (RuntimeArgumentHandle arglist);
```

Parameters

`arglist` RuntimeArgumentHandle

An argument list consisting of mandatory and optional arguments.

Remarks

The new ArgIterator object enumerates the argument list starting from the first optional argument.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

ArgIterator(RuntimeArgumentHandle, Void*)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the ArgIterator structure using the specified argument list and a pointer to an item in the list.

C#

```
[System.CLSCompliant(false)]
public ArgIterator (RuntimeArgumentHandle arglist, void* ptr);
```

Parameters

`arglist` RuntimeArgumentHandle

An argument list consisting of mandatory and optional arguments.

`ptr` Void*

A pointer to the argument in `arglist` to access first, or the first mandatory argument in `arglist` if `ptr` is `null`.

Attributes CLSCompliantAttribute

Remarks

The new ArgIterator object enumerates the argument list starting from the argument specified by `ptr`, or the first mandatory argument if `ptr` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgIterator.End Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Concludes processing of the variable-length argument list represented by this instance.

C#

```
public void End ();
```

Remarks

Conceptually, End concludes the processing of a variable-length argument list. However, this method has no actual implementation; the End method body is empty. You can use this method in your code as a marker to indicate where variable-length argument list processing logically ends.

The End method corresponds to the `va_end` method in the C standard library.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ArgIterator.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This method is not supported, and always throws NotSupportedException.

C#

```
public override bool Equals (object? o);
```

Parameters

o Object

An object to be compared to this instance.

Returns

Boolean

This comparison is not supported. No value is returned.

Exceptions

NotSupportedException

This method is not supported.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgIterator.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code of this object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgIterator.GetNextArg Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the next argument in a variable-length argument list.

Overloads

Expand table

GetNextArg()	Returns the next argument in a variable-length argument list.
GetNextArg(RuntimeTypeHandle)	Returns the next argument in a variable-length argument list that has a specified type.

GetNextArg()

> Important
>
> This API is not CLS-compliant.

Returns the next argument in a variable-length argument list.

C#

```
[System.CLSCompliant(false)]
public TypedReference GetNextArg ();
```

Returns

TypedReference

The next argument as a TypedReference object.

Attributes CLSCompliantAttribute

Exceptions

InvalidOperationException

An attempt was made to read beyond the end of the list.

Remarks

The iterator is automatically advanced to the next argument.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

GetNextArg(RuntimeTypeHandle)

> Important
>
> This API is not CLS-compliant.

Returns the next argument in a variable-length argument list that has a specified type.

C#

```
[System.CLSCompliant(false)]
public TypedReference GetNextArg (RuntimeTypeHandle rth);
```

Parameters

`rth` RuntimeTypeHandle

A runtime type handle that identifies the type of the argument to retrieve.

Returns

TypedReference

The next argument as a TypedReference object.

Attributes CLSCompliantAttribute

Exceptions

InvalidOperationException

An attempt was made to read beyond the end of the list.

ArgumentNullException

The pointer to the remaining arguments is zero.

Remarks

The iterator is automatically advanced to the next argument.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgIterator.GetNextArgType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the type of the next argument.

C#

```
public RuntimeTypeHandle GetNextArgType ();
```

Returns

RuntimeTypeHandle

The type of the next argument.

Remarks

This method does not advance the iterator to the next argument.

GetNextArgType returns the type of the argument as specified in the calling function. For example, if an argument is type String, but the calling function specifies the argument as type Object, GetNextArgType reports the argument as type Object.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgIterator.GetRemainingCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of arguments remaining in the argument list.

C#

```
public int GetRemainingCount ();
```

Returns

Int32

The number of remaining arguments.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when one of the arguments provided to a method is not valid.

C#

```
public class ArgumentException : SystemException
```

Inheritance Object → Exception → SystemException → ArgumentException

Derived System.ArgumentNullException
System.ArgumentOutOfRangeException
System.ComponentModel.InvalidAsynchronousStateException
System.ComponentModel.InvalidEnumArgumentException
System.DuplicateWaitObjectException
More...

Examples

The following example demonstrates how to throw and catch an ArgumentException. It uses the ArgumentException.GetType().Name property to display the name of the exception object, and also uses the Message property to display the text of the exception message.

C#

```
using System;

public class Example
{
    static void Main()
    {
        // Define some integers for a division operation.
        int[] values = { 10, 7 };
        foreach (var value in values) {
```

```
            try {
               Console.WriteLine("{0} divided by 2 is {1}", value,
DivideByTwo(value));
            }
            catch (ArgumentException e) {
               Console.WriteLine("{0}: {1}", e.GetType().Name, e.Message);
            }
            Console.WriteLine();
        }
    }

    static int DivideByTwo(int num)
    {
        // If num is an odd number, throw an ArgumentException.
        if ((num & 1) == 1)
            throw new ArgumentException(String.Format("{0} is not an even
number", num),
                                        "num");

        // num is even, return half of its value.
        return num / 2;
    }
}
// This example displays the following output:
//     10 divided by 2 is 5
//
//     ArgumentException: 7 is not an even number
//     Parameter name: num
```

Remarks

ArgumentException is thrown when a method is invoked and at least one of the passed arguments does not meet the parameter specification of the called method. The ParamName property identifies the invalid argument.

Most commonly, an ArgumentException is thrown by the common language runtime or another class library and indicates developer error. If you throw an ArgumentException from your code, you should ensure that the exception's Message property includes a meaningful error message that describes the invalid argument and the expected range of values for the argument.

The primary derived classes of ArgumentException are ArgumentNullException and ArgumentOutOfRangeException. These derived classes should be used instead of ArgumentException, except in situations where neither of the derived classes is acceptable. For example, exceptions should be thrown by:

- ArgumentNullException whenever `null` is passed to a method that does not accept it as a valid argument.

- ArgumentOutOfRangeException when the value of an argument is outside the range of acceptable values; for example, when the value "46" is passed as the month argument during the creation of a DateTime.

If the method call does not have any argument or if the failure does not involve the arguments themselves, then InvalidOperationException should be used.

ArgumentException uses the HRESULT COR_E_ARGUMENT, which has the value 0x80070057.

For a list of initial property values for an instance of ArgumentException, see the ArgumentException constructors.

In F#, you can use the invalidArg function to generate and raise an ArgumentException.

Constructors

Expand table

ArgumentException()	Initializes a new instance of the ArgumentException class.
ArgumentException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArgumentException class with serialized data.
ArgumentException(String)	Initializes a new instance of the ArgumentException class with a specified error message.
ArgumentException(String, Exception)	Initializes a new instance of the ArgumentException class with a specified error message and a reference to the inner exception that is the cause of this exception.
ArgumentException(String, String)	Initializes a new instance of the ArgumentException class with a specified error message and the name of the parameter that causes this exception.
ArgumentException(String, String, Exception)	Initializes a new instance of the ArgumentException class with a specified error message, the parameter name, and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message and the parameter name, or only the error message if no parameter name is set.
ParamName	Gets the name of the parameter that causes this exception.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the parameter name and additional exception information.
GetType()	Gets the runtime type of the current instance.

	(Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ThrowIfNullOrEmpty(String, String)	Throws an exception if `argument` is `null` or empty.
ThrowIfNullOrWhiteSpace(String, String)	Throws an exception if `argument` is `null`, empty, or consists only of white-space characters.
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArgumentException class.

Overloads

Expand table

ArgumentException()	Initializes a new instance of the ArgumentException class.
ArgumentException(String)	Initializes a new instance of the ArgumentException class with a specified error message.
ArgumentException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArgumentException class with serialized data.
ArgumentException(String, Exception)	Initializes a new instance of the ArgumentException class with a specified error message and a reference to the inner exception that is the cause of this exception.
ArgumentException(String, String)	Initializes a new instance of the ArgumentException class with a specified error message and the name of the parameter that causes this exception.
ArgumentException(String, String, Exception)	Initializes a new instance of the ArgumentException class with a specified error message, the parameter name, and a reference to the inner exception that is the cause of this exception.

ArgumentException()

Initializes a new instance of the ArgumentException class.

```
C#
```

```
public ArgumentException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "An invalid argument was specified." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ArgumentException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentException(String)

Initializes a new instance of the ArgumentException class with a specified error message.

C#

```
public ArgumentException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "An invalid argument was specified." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ArgumentException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ArgumentException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ArgumentException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ArgumentException(String, Exception)

Initializes a new instance of the ArgumentException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public ArgumentException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

This constructor initializes the Message property of the new instance using the value of the `message` parameter. The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ArgumentException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The localized error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentException(String, String)

Initializes a new instance of the ArgumentException class with a specified error message and the name of the parameter that causes this exception.

C#

```
public ArgumentException (string? message, string? paramName);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`paramName` String

The name of the parameter that caused the current exception.

Examples

The following code example demonstrates how to call the ArgumentException constructor. This code example is part of a larger example provided for the ArgumentException class.

C#

```
static int DivideByTwo(int num)
{
    // If num is an odd number, throw an ArgumentException.
    if ((num & 1) == 1)
        throw new ArgumentException("Number must be even", "num");

    // num is even, return half of its value.
    return num / 2;
}
```

Remarks

This constructor initializes the Message property of the new instance using the value of the `message` parameter. The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This constructor initializes the ParamName property of the new instance using `paramName`. The content of `paramName` is intended to be understood by humans.

The following table shows the initial property values for an instance of ArgumentException.

Expand table

Property	Value
Message	The error message string.
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentException(String, String, Exception)

Initializes a new instance of the ArgumentException class with a specified error message, the parameter name, and a reference to the inner exception that is the cause of this exception.

C#

```
public ArgumentException (string? message, string? paramName, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`paramName` String

The name of the parameter that caused the current exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

This constructor initializes the Message property of the new instance using the value of the `message` parameter. The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This constructor initializes the ParamName property of the new instance using `paramName`. The content of `paramName` is intended to be understood by humans.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ArgumentException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The localized error message string.
ParamName	The parameter name string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the error message and the parameter name, or only the error message if no parameter name is set.

C#

```
public override string Message { get; }
```

Property Value

String

A text string describing the details of the exception. The value of this property takes one of two forms:

Expand table

Condition	Value
The `paramName` is a null reference (`Nothing` in Visual Basic) or of zero length.	The `message` string passed to the constructor.
The `paramName` is not null reference (`Nothing` in Visual Basic) and it has a length greater than zero.	The `message` string appended with the name of the invalid parameter.

Remarks

This property overrides Message. The error message should be localized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentException.ParamName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the parameter that causes this exception.

```
C#
public virtual string? ParamName { get; }
```

Property Value

String

The parameter name.

Remarks

Every ArgumentException should carry the name of the parameter that causes this exception.

This property is read-only, and returns the same value as was passed into the constructor. Overriding methods should be used solely to customize the format of the parameter name.

The parameter name should not be translated for localization purposes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentException.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the parameter name and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is a null reference (`Nothing` in Visual Basic).

Remarks

GetObjectData sets a SerializationInfo with all the exception object data targeted for serialization. During deserialization, the exception object is reconstituted from the SerializationInfo transmitted over the stream.

For more information, see SerializationInfo.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- XML and SOAP Serialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentException.ThrowIfNullOrEmpty(String, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an exception if `argument` is `null` or empty.

```
C#

public static void ThrowIfNullOrEmpty (string? argument, string? paramName =
default);
```

Parameters

`argument` String

The string argument to validate as non-`null` and non-empty.

`paramName` String

The name of the parameter with which `argument` corresponds.

Exceptions

ArgumentNullException

`argument` is `null`.

ArgumentException

`argument` is empty.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentException.ThrowIfNullOrWhiteSpace(String, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an exception if `argument` is `null`, empty, or consists only of white-space characters.

```
C#

public static void ThrowIfNullOrWhiteSpace (string? argument, string?
paramName = default);
```

Parameters

`argument` String

The string argument to validate.

`paramName` String

The name of the parameter with which `argument` corresponds.

Exceptions

ArgumentNullException

`argument` is `null`.

ArgumentException

`argument` is empty or consists only of white-space characters.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentNullException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a null reference (`Nothing` in Visual Basic) is passed to a method that does not accept it as a valid argument.

C#

```
public class ArgumentNullException : ArgumentException
```

Inheritance Object → Exception → SystemException → ArgumentException → ArgumentNullException

Remarks

An ArgumentNullException exception is thrown when a method is invoked and at least one of the passed arguments is `null` but should never be `null`.

An ArgumentNullException exception is thrown at run time in the following two major circumstances, both of which reflect developer error:

- An uninstantiated object is passed to a method. To prevent the error, instantiate the object.
- An object returned from a method call is then passed as an argument to a second method, but the value of the original returned object is `null`. To prevent the error, check for a return value that is `null` and call the second method only if the return value is not `null`.

ArgumentNullException behaves identically to ArgumentException. It is provided so that application code can differentiate between exceptions caused by `null` arguments and exceptions caused by arguments that are not null. For errors caused by arguments that are not null, see ArgumentOutOfRangeException.

ArgumentNullException uses the HRESULT E_POINTER, which has the value 0x80004003.

For a list of initial property values for an instance of ArgumentNullException, see the ArgumentNullException constructors.

Constructors

Expand table

ArgumentNullException()	Initializes a new instance of the ArgumentNullException class.
ArgumentNullException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArgumentNullException class with serialized data.
ArgumentNullException(String)	Initializes a new instance of the ArgumentNullException class with the name of the parameter that causes this exception.
ArgumentNullException(String, Exception)	Initializes a new instance of the ArgumentNullException class with a specified error message and the exception that is the cause of this exception.
ArgumentNullException(String, String)	Initializes an instance of the ArgumentNullException class with a specified error message and the name of the parameter that causes this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message and the parameter name, or only the error message if no parameter name is set.

	(Inherited from ArgumentException)
ParamName	Gets the name of the parameter that causes this exception. (Inherited from ArgumentException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the parameter name and additional exception information. (Inherited from ArgumentException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ThrowIfNull(Object, String)	Throws an ArgumentNullException if `argument` is `null`.
ThrowIfNull(Void*, String)	Throws an ArgumentNullException if `argument` is `null`.
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentNullException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArgumentNullException class.

Overloads

Expand table

ArgumentNullException()	Initializes a new instance of the ArgumentNullException class.
ArgumentNullException(String)	Initializes a new instance of the ArgumentNullException class with the name of the parameter that causes this exception.
ArgumentNullException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArgumentNullException class with serialized data.
ArgumentNullException(String, Exception)	Initializes a new instance of the ArgumentNullException class with a specified error message and the exception that is the cause of this exception.
ArgumentNullException(String, String)	Initializes an instance of the ArgumentNullException class with a specified error message and the name of the parameter that causes this exception.

ArgumentNullException()

Initializes a new instance of the ArgumentNullException class.

```
C#
```

```
public ArgumentNullException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Value cannot be null." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ArgumentNullException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	A localized error message string, such as "Value cannot be null." for the English language.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentNullException(String)

Initializes a new instance of the ArgumentNullException class with the name of the parameter that causes this exception.

C#

```
public ArgumentNullException (string? paramName);
```

Parameters

paramName String

The name of the parameter that caused the exception.

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error and includes the parameter name. This message takes into account the current system culture.

This constructor initializes the ParamName property of the new instance using the `paramName` parameter. The content of `paramName` is intended to be understood by humans.

The following table shows the initial property values for an instance of ArgumentNullException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	A localized error message string that identifies the null argument. For example, if the `paramName` argument is "arg1", the English language message string is: On .NET 5+ and .NET Core: `Value cannot be null. (Parameter name: 'arg1')` On .NET Framework: `Value cannot be null.\r\nParameter name: arg1`
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentNullException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ArgumentNullException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ArgumentNullException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

An object that describes the source or destination of the serialized data.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ArgumentNullException(String, Exception)

Initializes a new instance of the ArgumentNullException class with a specified error message and the exception that is the cause of this exception.

C#

```
public ArgumentNullException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for this exception.

`innerException` Exception

The exception that is the cause of the current exception, or a null reference (`Nothing` in Visual Basic) if no inner exception is specified.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentNullException(String, String)

Initializes an instance of the ArgumentNullException class with a specified error message and the name of the parameter that causes this exception.

C#

```
public ArgumentNullException (string? paramName, string? message);
```

Parameters

`paramName` String

The name of the parameter that caused the exception.

`message` String

A message that describes the error.

Remarks

This constructor initializes the Message property of the new instance using the value of the `message` parameter. The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This constructor initializes the ParamName property of the new instance using the `paramName` parameter. The content of `paramName` is intended to be understood by

humans.

The following table shows the initial property values for an instance of ArgumentNullException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentNullException.ThrowIfNull Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ThrowIfNull(Object, String)	Throws an ArgumentNullException if `argument` is `null`.
ThrowIfNull(Void*, String)	Throws an ArgumentNullException if `argument` is `null`.

ThrowIfNull(Object, String)

Throws an ArgumentNullException if `argument` is `null`.

C#

```
public static void ThrowIfNull (object? argument, string? paramName =
default);
```

Parameters

`argument` Object

The reference type argument to validate as non-null.

`paramName` String

The name of the parameter with which `argument` corresponds. If you omit this parameter, the name of `argument` is used.

Exceptions

ArgumentNullException

`argument` is `null`.

Remarks

The `paramName` parameter is included to support the CallerArgumentExpressionAttribute attribute. It's recommended that you don't pass a value for this parameter and let the name of `argument` be used instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ThrowIfNull(Void*, String)

> Important
>
> This API is not CLS-compliant.

Throws an ArgumentNullException if `argument` is `null`.

C#

```
[System.CLSCompliant(false)]
public static void ThrowIfNull (void* argument, string? paramName =
default);
```

Parameters

`argument` Void*

The pointer argument to validate as non-null.

`paramName` String

The name of the parameter with which `argument` corresponds.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.

C#

```
public class ArgumentOutOfRangeException : ArgumentException
```

Inheritance Object → Exception → SystemException → ArgumentException → ArgumentOutOfRangeException

Examples

The following example defines a class to contain information about an invited guest. If the guest is younger than 21, an ArgumentOutOfRangeException exception is thrown.

C#

```
using System;
using static System.Console;

public class Program
{
    public static void Main(string[] args)
    {
        try
        {
            var guest1 = new Guest("Ben", "Miller", 17);
            WriteLine(guest1.GuestInfo);
        }
        catch (ArgumentOutOfRangeException argumentOutOfRangeException)
        {
            WriteLine($"Error: {argumentOutOfRangeException.Message}");
        }
    }
}
```

```
class Guest
{
    private const int minimumRequiredAge = 21;

    private string firstName;
    private string lastName;
    private int age;

    public Guest(string firstName, string lastName, int age)
    {
        if (age < minimumRequiredAge)
            throw new ArgumentOutOfRangeException(nameof(age), $"All guests
must be {minimumRequiredAge}-years-old or older.");

        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public string GuestInfo => $"{firstName} {lastName}, {age}";
}
```

Remarks

An ArgumentOutOfRangeException exception is thrown when a method is invoked and at least one of the arguments passed to the method is not `null` and contains an invalid value that is not a member of the set of values expected for the argument. The ParamName property identifies the invalid argument, and the ActualValue property, if a value is present, identifies the invalid value.

Typically, an ArgumentOutOfRangeException results from developer error. Instead of handling the exception in a `try`/`catch` block, you should eliminate the cause of the exception or, if the argument is returned by a method call or input by the user before being passed to the method that throws the exception, you should validate arguments before passing them to the method.

ArgumentOutOfRangeException is used extensively by:

- Classes in the System.Collections and System.IO namespaces.
- The Array class.
- String manipulation methods in the String class.

The conditions in which an ArgumentOutOfRangeException exception is thrown include the following:

- You are retrieving the member of a collection by its index number, and the index number is invalid.

  This is the most common cause of an ArgumentOutOfRangeException exception. Typically, the index number is invalid for one of four reasons:

  1. The collection has no members, and your code assumes that it does. The following example attempts to retrieve the first element of a collection that has no elements:

     C#

     ```
     using System;
     using System.Collections.Generic;

     public class Example4
     {
        public static void Main()
        {
           var list = new List<string>();
           Console.WriteLine("Number of items: {0}", list.Count);
           try {
              Console.WriteLine("The first item: '{0}'", list[0]);
           }
           catch (ArgumentOutOfRangeException e) {
              Console.WriteLine(e.Message);
           }
        }
     }
     // The example displays the following output:
     //   Number of items: 0
     //   Index was out of range. Must be non-negative and less than
     the size of the collection.
     //   Parameter name: index
     ```

     To prevent the exception, check whether the collection's `Count` property is greater than zero before attempting to retrieve any members, as the following code fragment does.

     C#

     ```
     if (list.Count > 0)
        Console.WriteLine("The first item: '{0}'", list[0]);
     ```

  2. In some cases, the exception may occur because you are attempting to add a member to a collection by using an index that does not exist, rather than by calling the method, such as `Add`, that exists for this purpose. The following

example attempts to add an element to a collection by using a non-existent index rather than calling the List<T>.Add method.

```
C#

using System;
using System.Collections.Generic;

public class Example13
{
   public static void Main()
   {
      var numbers = new List<int>();
      numbers.AddRange( new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
20 } );

      var squares = new List<int>();
      for (int ctr = 0; ctr < numbers.Count; ctr++)
         squares[ctr] = (int) Math.Pow(numbers[ctr], 2);
   }
}
// The example displays the following output:
//    Unhandled Exception: System.ArgumentOutOfRangeException:
Index was out of range. Must be non-negative and less than the
size of the collection.
//    Parameter name: index
//       at
System.ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgum
ent argument, ExceptionResource resource)
//       at Example.Main()
```

The following code fragment corrects this error:

```
C#

var squares = new List<int>();
for (int ctr = 0; ctr < numbers.Count; ctr++)
   squares.Add((int) Math.Pow(numbers[ctr], 2));
```

3. You're attempting to retrieve an item whose index is negative. This usually occurs because you've searched a collection for the index of a particular element and have erroneously assumed that the search is successful. In the following example, the call to the List<T>.FindIndex(Predicate<T>) method fails to find a string equal to "Z" and so returns -1. However, this is an invalid index value.

```
C#
```

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      var list = new List<string>();
      list.AddRange( new String[] { "A", "B", "C" } );
      // Get the index of the element whose value is "Z".
      int index = list.FindIndex((new
StringSearcher("Z")).FindEquals);
      try {
         Console.WriteLine("Index {0} contains '{1}'", index,
list[index]);
      }
      catch (ArgumentOutOfRangeException e) {
         Console.WriteLine(e.Message);
      }
   }
}

internal class StringSearcher
{
   string value;

   public StringSearcher(string value)
   {
      this.value = value;
   }

   public bool FindEquals(string s)
   {
      return s.Equals(value, StringComparison.InvariantCulture);
   }
}
// The example displays the following output:
//   Index was out of range. Must be non-negative and less than
the size of the collection.
//   Parameter name: index
```

To prevent the exception, check that the search is successful by making sure that the returned index is greater than or equal to zero before attempting to retrieve the item from the collection, as the following code fragment does.

C#

```
// Get the index of the element whose value is "Z".
int index = list.FindIndex((new StringSearcher("Z")).FindEquals);
if (index >= 0)
   Console.WriteLine("'Z' is found at index {0}", list[index]);
```

4. You're attempting to retrieve an element whose index is equal to the value of the collection's `Count` property, as the following example illustrates.

C#

```
using System;
using System.Collections.Generic;

public class Example8
{
   public static void Main()
   {
      var list = new List<string>();
      list.AddRange( new String[] { "A", "B", "C" } );
      try {
         // Display the elements in the list by index.
         for (int ctr = 0; ctr <= list.Count; ctr++)
            Console.WriteLine("Index {0}: {1}", ctr, list[ctr]);
      }
      catch (ArgumentOutOfRangeException e) {
         Console.WriteLine(e.Message);
      }
   }
}
// The example displays the following output:
//   Index 0: A
//   Index 1: B
//   Index 2: C
//   Index was out of range. Must be non-negative and less than
the size of the collection.
//   Parameter name: index
```

Because collections in .NET use zero-based indexing, the first element of the collection is at index 0, and the last element is at index `Count` - 1. You can eliminate the error by ensuring that you access the last element at index `Count` - 1, as the following code does.

C#

```
// Display the elements in the list by index.
for (int ctr = 0; ctr < list.Count; ctr++)
   Console.WriteLine("Index {0}: {1}", ctr, list[ctr]);
```

- You are attempting to perform a string operation by calling a string manipulation method, and the starting index does not exist in the string.

  Overloads of methods such as such as String.Compare, String.CompareOrdinal, String.IndexOf, IndexOfAny, String.Insert, String.LastIndexOf, String.LastIndexOfAny,

Remove, or String.Substring that allow you to specify the starting index of the operation require that the index be a valid position within the string. Valid indexes range from 0 to String.Length - 1.

There are four common causes of this ArgumentOutOfRangeException exception:

1. You are working with an empty string, or String.Empty. Because its String.Length property returns 0, any attempt to manipulate it by index throws an ArgumentOutOfRangeException exception. The following example, defines a `GetFirstCharacter` method that returns the first character of a string. If the string is empty, as the final string passed to the method is, the method throws an ArgumentOutOfRangeException exception.

   C#

   ```
   using System;

   public class Example1
   {
       public static void Main()
       {
           String[] words = { "the", "today", "tomorrow", " ", "" };
           foreach (var word in words)
               Console.WriteLine("First character of '{0}': '{1}'",
                                 word, GetFirstCharacter(word));
       }

       private static char GetFirstCharacter(string s)
       {
           return s[0];
       }
   }
   // The example displays the following output:
   //    First character of //the//: //t//
   //    First character of //today//: //t//
   //    First character of //tomorrow//: //t//
   //    First character of // //: // //
   //
   //    Unhandled Exception: System.IndexOutOfRangeException: Index
   was outside the bounds of the array.
   //       at Example.Main()
   ```

   You can eliminate the exception by testing whether the string's String.Length is greater than zero or by calling the IsNullOrEmpty method to ensure that the string is not `null` or empty. The following code fragment does the latter. In this case, if the string is `null` or empty, the `GetFirstCharacter` method returns U+0000.

C#

```
static char GetFirstCharacter(string s)
{
    if (string.IsNullOrEmpty(s))
        return '\u0000';
    else
        return s[0];
}
```

2. You're manipulating a string based on the position of a substring within that string, and you've failed to determine whether the substring was actually found.

   The following example extracts the second word of a two-word phrase. It throws an ArgumentOutOfRangeException exception if the phrase consists of only one word, and therefore does not contain an embedded space character. This occurs because the call to the String.IndexOf(String) method returns -1 to indicate that the search failed, and this invalid value is then passed to the String.Substring(Int32) method.

C#

```
using System;

public class Example17
{
   public static void Main()
   {
      String[] phrases = { "ocean blue", "concerned citizen",
                           "runOnPhrase" };
      foreach (var phrase in phrases)
         Console.WriteLine("Second word is {0}",
GetSecondWord(phrase));
   }

   static string GetSecondWord(string s)
   {
      int pos = s.IndexOf(" ");
      return s.Substring(pos).Trim();
   }
}
// The example displays the following output:
//    Second word is blue
//    Second word is citizen
//
//    Unhandled Exception: System.ArgumentOutOfRangeException:
StartIndex cannot be less than zero.
//    Parameter name: startIndex
//       at System.String.Substring(Int32 startIndex, Int32
```

```
length)
//       at Example17.GetSecondWord(String s)
//       at Example17.Main()
```

To eliminate the exception, validate the value returned by the string search method before calling the string manipulation method.

C#

```
using System;

public class Example18
{
   public static void Main()
   {
      String[] phrases = { "ocean blue", "concerned citizen",
                           "runOnPhrase" };
      foreach (var phrase in phrases) {
         string word = GetSecondWord(phrase);
         if (! string.IsNullOrEmpty(word))
            Console.WriteLine("Second word is {0}", word);
      }
   }

   static string GetSecondWord(string s)
   {
      int pos = s.IndexOf(" ");
      if (pos >= 0)
         return s.Substring(pos).Trim();
      else
         return string.Empty;
   }
}
// The example displays the following output:
//       Second word is blue
//       Second word is citizen
```

3. You've attempted to extract a substring that is outside the range of the current string.

   The methods that extract substrings all require that you specify the starting position of the substring and, for substrings that do not continue to the end of the string, the number of characters in the substring. Note that this is not the *index* of the last character in the substring.

   An ArgumentOutOfRangeException exception is typically thrown in this case because you've incorrectly calculated the number of characters in the substring. If you are using a search method like String.IndexOf to identify the starting and ending positions of a substring:

- If the character in the ending position returned by String.IndexOf is to be included in the substring, the ending position of the substring is given by the formula

```
endIndex - startIndex + 1
```

- If the character in the ending position returned by String.IndexOf is to be excluded from the substring, the ending position of the substring is given by the formula

```
endIndex - startIndex
```

The following example defines a `FindWords` method that uses the String.IndexOfAny(Char[], Int32) method to identify space characters and punctuation marks in a string and returns an array that contains the words found in the string.

C#

```
using System;
using System.Collections.Generic;

public class Example19
{
   public static void Main()
   {
      string sentence = "This is a simple, short sentence.";
      Console.WriteLine("Words in '{0}':", sentence);
      foreach (var word in FindWords(sentence))
         Console.WriteLine("   '{0}'", word);
   }

   static String[] FindWords(string s)
   {
      int start = 0, end = 0;
      Char[] delimiters = { ' ', '.', ',', ';', ':', '(', ')'
};
      var words = new List<string>();

      while (end >= 0) {
         end = s.IndexOfAny(delimiters, start);
         if (end >= 0) {
            if (end - start > 0)
               words.Add(s.Substring(start, end - start));
```

```
            start = end + 1;
         }
         else {
            if (start < s.Length - 1)
               words.Add(s.Substring(start));
         }
      }
      return words.ToArray();
   }
}
// The example displays the following output:
//       Words in 'This is a simple, short sentence.':
//          'This'
//          'is'
//          'a'
//          'simple'
//          'short'
//          'sentence'
```

- You have passed a negative number to a method with an argument that requires only positive numbers and zero, or you have passed either a negative number or zero to a method with an argument that requires only positive numbers.

  For example, the Array.CreateInstance(Type, Int32, Int32, Int32) method requires that you specify the number of elements in each dimension of a two-dimensional array; valid values for each dimension can range from 0 to Int32.MaxValue. But because the dimension argument in the following example has a negative value, the method throws an ArgumentOutOfRangeException exception.

C#

```
using System;

public class Example01
{
    public static void Main()
    {
        int dimension1 = 10;
        int dimension2 = -1;
        try
        {
            Array arr = Array.CreateInstance(typeof(string),
                                             dimension1, dimension2);
        }
        catch (ArgumentOutOfRangeException e)
        {
            if (e.ActualValue != null)
                Console.WriteLine("{0} is an invalid value for {1}: ",
e.ActualValue, e.ParamName);
            Console.WriteLine(e.Message);
```

```
        }
    }
}
// The example displays the following output:
//     Non-negative number required.
//     Parameter name: length2
```

To correct the error, ensure that the value of the invalid argument is non-negative. You can do this by providing a valid value, as the following code fragment does.

```C#
int dimension1 = 10;
int dimension2 = 10;
Array arr = Array.CreateInstance(typeof(string),
                                 dimension1, dimension2);
```

You can also validate the input and, if it is invalid, take some action. The following code fragment displays an error message instead of calling the method.

```C#
if (dimension1 < 0 || dimension2 < 0)
{
    Console.WriteLine("Unable to create the array.");
    Console.WriteLine("Specify non-negative values for the two
dimensions.");
}
else
{
    arr = Array.CreateInstance(typeof(string),
                               dimension1, dimension2);
}
```

- A race condition exists in an app that is multithreaded or has tasks that execute asynchronously and that updates an array or collection.

  The following example uses a List<T> object to populate a collection of `Continent` objects. It throws an ArgumentOutOfRangeException if the example attempts to display the seven items in the collection before the collection is fully populated.

```C#
using System;
using System.Collections.Generic;
using System.Threading;

public class Continent
```

```
{
   public string? Name { get; set; }
   public int Population { get; set; }
   public Decimal Area { get; set; }
}

public class Example11
{
   static List<Continent> continents = new List<Continent>();
   static string? s_msg;

   public static void Main()
   {
      String[] names = { "Africa", "Antarctica", "Asia",
                         "Australia", "Europe", "North America",
                         "South America" };
      // Populate the list.
      foreach (var name in names) {
         var th = new Thread(PopulateContinents);
         th.Start(name);
      }
      Console.WriteLine(s_msg);
      Console.WriteLine();

      // Display the list.
      for (int ctr = 0; ctr < names.Length; ctr++) {
         var continent = continents[ctr];
         Console.WriteLine("{0}: Area: {1}, Population {2}",
                           continent.Name, continent.Population,
                           continent.Area);
      }
   }

   private static void PopulateContinents(Object? obj)
   {
      string? name = obj?.ToString();
      s_msg += string.Format("Adding '{0}' to the list.\n", name);
      var continent = new Continent();
      continent.Name = name;
      // Sleep to simulate retrieving remaining data.
      Thread.Sleep(50);
      continents.Add(continent);
   }
}
// The example displays output like the following:
//    Adding //Africa// to the list.
//    Adding //Antarctica// to the list.
//    Adding //Asia// to the list.
//    Adding //Australia// to the list.
//    Adding //Europe// to the list.
//    Adding //North America// to the list.
//    Adding //South America// to the list.
//
//
//
```

```
//    Unhandled Exception: System.ArgumentOutOfRangeException: Index
was out of range. Must be non-negative and less than the size of the
collection.
//    Parameter name: index
//       at
System.ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgument
argument, ExceptionResource resource)
//       at Example.Main()
```

In this case, two resources are accessed from multiple threads:

- The `continents` collection. Its List<T>.Add method is called from multiple threads. In addition, the main or primary thread assumes the collection is fully populated with seven elements when it iterates its members.
- The `msg` string, which is concatenated from multiple threads.

To correct the error, ensure that shared state is accessed in a thread-safe way, as follows.

- if your app uses an array or collection object, consider using a thread-safe collection class, such as the types in the System.Collections.Concurrent namespace or the System.Collections.Immutable out-of-band release.
- Ensure that shared state (that is, resources that can be accessed by multiple threads) is accessed in a thread-safe way, so that only one thread at a time has exclusive access to the resources. A large number of classes, such as CountdownEvent, Interlocked, Monitor, and Mutex, are available to synchronize access to resources. For more information, see Threading. In addition, language support is available through the lock statement in C# and the SyncLock construct in Visual Basic.

The following example addresses the ArgumentOutOfRangeException and the other issues from the previous example. It replaces the List<T> object with a ConcurrentBag<T> object to ensure that access to the collection is thread-safe, uses a CountdownEvent object to ensure that the application thread continues only after other threads have executed, and uses a lock to ensure that only one thread can access the `msg` variable at a time.

C#

```
using System;
using System.Collections.Concurrent;
using System.Threading;

public class ContinentD
```

```
{
   public string? Name { get; set; }
   public int Population { get; set; }
   public Decimal Area { get; set; }
}

public class Example12
{
   static ConcurrentBag<ContinentD> ContinentDs = new
ConcurrentBag<ContinentD>();
   static CountdownEvent? gate;
   static string msg = string.Empty;

   public static void Main()
   {
      String[] names = { "Africa", "Antarctica", "Asia",
                         "Australia", "Europe", "North America",
                         "South America" };
      gate = new CountdownEvent(names.Length);

      // Populate the list.
      foreach (var name in names) {
         var th = new Thread(PopulateContinentDs);
         th.Start(name);
      }

      // Display the list.
      gate.Wait();
      Console.WriteLine(msg);
      Console.WriteLine();

      var arr = ContinentDs.ToArray();
      for (int ctr = 0; ctr < names.Length; ctr++) {
         var ContinentD = arr[ctr];
         Console.WriteLine("{0}: Area: {1}, Population {2}",
                           ContinentD.Name, ContinentD.Population,
                           ContinentD.Area);
      }
   }

   private static void PopulateContinentDs(Object? obj)
   {
      string? name = obj?.ToString();
      lock(msg) {
         msg += string.Format("Adding '{0}' to the list.\n", name);
      }
      var ContinentD = new ContinentD();
      ContinentD.Name = name;
      // Sleep to simulate retrieving remaining data.
      Thread.Sleep(25);
      ContinentDs.Add(ContinentD);
      gate?.Signal();
   }
}
// The example displays output like the following:
```

```
//       Adding 'Africa' to the list.
//       Adding 'Antarctica' to the list.
//       Adding 'Asia' to the list.
//       Adding 'Australia' to the list.
//       Adding 'Europe' to the list.
//       Adding 'North America' to the list.
//       Adding 'South America' to the list.
//
//
//       Africa: Area: 0, Population 0
//       Antarctica: Area: 0, Population 0
//       Asia: Area: 0, Population 0
//       Australia: Area: 0, Population 0
//       Europe: Area: 0, Population 0
//       North America: Area: 0, Population 0
//       South America: Area: 0, Population 0
```

ArgumentOutOfRangeException uses the HRESULT COR_E_ARGUMENTOUTOFRANGE, which has the value 0x80131502.

For a list of initial property values for an instance of ArgumentOutOfRangeException, see the ArgumentOutOfRangeException constructors.

Constructors

Expand table

ArgumentOutOfRangeException()	Initializes a new instance of the ArgumentOutOfRangeException class.
ArgumentOutOfRange Exception(SerializationInfo, Streaming Context)	**Obsolete.** Initializes a new instance of the ArgumentOutOfRangeException class with serialized data.
ArgumentOutOfRange Exception(String)	Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception.
ArgumentOutOfRange Exception(String, Exception)	Initializes a new instance of the ArgumentOutOfRangeException class with a specified error message and the exception that is the cause of this exception.
ArgumentOutOfRange Exception(String, Object, String)	Initializes a new instance of the ArgumentOutOfRangeException class with the parameter name, the value of the argument, and a specified error message.

ArgumentOutOfRangeException(String, String)	Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception and a specified error message.

Properties

Expand table

ActualValue	Gets the argument value that causes this exception.
Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message and the string representation of the invalid argument value, or only the error message if the argument value is null.
ParamName	Gets the name of the parameter that causes this exception. (Inherited from ArgumentException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.

	(Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the invalid argument value and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ThrowIfEqual<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is equal to `other`.
ThrowIfGreaterThan<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is greater than `other`.
ThrowIfGreaterThanOrEqual<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is greater than or equal to `other`.
ThrowIfLessThan<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is less than `other`.
ThrowIfLessThanOrEqual<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is less than or equal to `other`.
ThrowIfNegative<T>(T, String)	Throws an ArgumentOutOfRangeException if `value` is negative.
ThrowIfNegativeOrZero<T>(T, String)	Throws an ArgumentOutOfRangeException if `value` is negative or zero.
ThrowIfNotEqual<T>(T, T, String)	Throws an ArgumentOutOfRangeException if `value` is not equal to `other`.
ThrowIfZero<T>(T, String)	Throws an ArgumentOutOfRangeException if `value` is zero.
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArgumentOutOfRangeException class.

Overloads

Expand table

ArgumentOutOfRangeException()	Initializes a new instance of the ArgumentOutOfRangeException class.
ArgumentOutOfRangeException(String)	Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception.
ArgumentOutOfRangeException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArgumentOutOfRangeException class with serialized data.
ArgumentOutOfRangeException(String, Exception)	Initializes a new instance of the ArgumentOutOfRangeException class with a specified error message and the exception that is the cause of this exception.
ArgumentOutOfRangeException(String, String)	Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception and a specified error message.
ArgumentOutOfRangeException(String, Object, String)	Initializes a new instance of the ArgumentOutOfRangeException class with the parameter name, the value of the argument, and a specified error message.

ArgumentOutOfRangeException()

Initializes a new instance of the ArgumentOutOfRangeException class.

C#

```
public ArgumentOutOfRangeException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Nonnegative number required." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ArgumentOutOfRangeException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentOutOfRangeException(String)

Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception.

```
C#

public ArgumentOutOfRangeException (string? paramName);
```

Parameters

paramName String

The name of the parameter that causes this exception.

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Nonnegative number required." This message takes into account the current system culture.

This constructor initializes the ParamName property of the new instance using the `paramName` parameter. The content of `paramName` is intended to be understood by humans.

The following table shows the initial property values for an instance of ArgumentOutOfRangeException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The empty string ("").
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentOutOfRangeException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ArgumentOutOfRangeException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ArgumentOutOfRangeException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

An object that describes the source or destination of the serialized data.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ArgumentOutOfRangeException(String, Exception)

Initializes a new instance of the ArgumentOutOfRangeException class with a specified error message and the exception that is the cause of this exception.

C#

```
public ArgumentOutOfRangeException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for this exception.

`innerException` Exception

The exception that is the cause of the current exception, or a null reference (`Nothing` in Visual Basic) if no inner exception is specified.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentOutOfRangeException(String, String)

Initializes a new instance of the ArgumentOutOfRangeException class with the name of the parameter that causes this exception and a specified error message.

C#

```
public ArgumentOutOfRangeException (string? paramName, string? message);
```

Parameters

`paramName` String

The name of the parameter that caused the exception.

`message` String

The message that describes the error.

Remarks

This constructor initializes the Message property of the new instance using the value of the `message` parameter. The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This constructor initializes the ParamName property of the new instance using the `paramName` parameter. The content of `paramName` is intended to be understood by

humans.

The following table shows the initial property values for an instance of ArgumentOutOfRangeException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArgumentOutOfRangeException(String, Object, String)

Initializes a new instance of the ArgumentOutOfRangeException class with the parameter name, the value of the argument, and a specified error message.

C#

```
public ArgumentOutOfRangeException (string? paramName, object?
actualValue, string? message);
```

Parameters

`paramName` String

The name of the parameter that caused the exception.

`actualValue` Object

The value of the argument that causes this exception.

`message` String

The message that describes the error.

Remarks

This constructor initializes the ParamName property of the new instance using the `paramName` parameter, the ActualValue property using the `actualValue` parameter, and the Message property using the `message` parameter. The content of the `paramName` and `message` parameters is intended to be understood by humans. The caller of this constructor is required to ensure that these strings have been localized for the current system culture.

The `actualValue` parameter is not used within the .NET class library. However, apps can use the ActualValue property to obtain the available argument value.

The `actualValue` parameter contains the invalid value that was passed to the method and caused this exception to be thrown. This value is stored in the ActualValue property and its string representation is appended to the message string held in the Message property.

The following table shows the initial property values for an instance of ArgumentOutOfRangeException.

Expand table

Property	Value
ActualValue	The argument value.
InnerException	The null reference (`Nothing` in Visual Basic).
Message	The error message string.
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException.ActualValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the argument value that causes this exception.

```
C#
public virtual object? ActualValue { get; }
```

Property Value

Object

The value of the parameter that caused the current Exception.

Remarks

Use this property to retrieve the invalid argument.

The `ActualValue` property returns the same value that was passed to the constructor, or `null` if no value was supplied.

The ActualValue property is assigned a value at the time of object construction. If the ActualValue property value is not `null`, a string representation of the value is appended to the message string held by the Message property.

The ActualValue property is not used within the .NET class library. It carries a null value in all the ArgumentOutOfRangeException objects thrown by the .NET class library. The ActualValue property is provided so that applications can use the available argument value.

This property is read-only and should be overridden only to customize the content or format of the value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the error message and the string representation of the invalid argument value, or only the error message if the argument value is null.

C#

```
public override string Message { get; }
```

Property Value

String

The text message for this exception. The value of this property takes one of two forms, as follows.

Expand table

Condition	Value
The `actualValue` is `null`.	The `message` string passed to the constructor.
The `actualValue` is not `null`.	The `message` string appended with the string representation of the invalid argument value.

Remarks

This property overrides Message.

The error message should describe the expected values of the parameter that causes this exception. The error message should be localized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the invalid argument value and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

An object that describes the source or destination of the serialized data.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is `null`.

Remarks

GetObjectData sets a SerializationInfo object with all the exception object data targeted for serialization. During deserialization, the exception object is reconstituted from the SerializationInfo transmitted over the stream.

For more information, see SerializationInfo.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArgumentOutOfRangeException.ThrowIfEqual<T>(T, T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is equal to `other`.

```
C#

public static void ThrowIfEqual<T> (T value, T other, string? paramName =
default) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the objects to validate.

Parameters

`value` T

The argument to validate as not equal to `other`.

`other` T

The value to compare with `value`.

`paramName` String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfGreaterThan<T>(T, T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is greater than `other`.

```
C#

public static void ThrowIfGreaterThan<T> (T value, T other, string?
paramName = default) where T : IComparable<T>;
```

Type Parameters

`T`

The type of the objects to validate.

Parameters

`value` T

The argument to validate as less than or equal to `other`.

`other` T

The value to compare with `value`.

`paramName` String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfGreaterThanOrEqual<T> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is greater than or equal to `other`.

C#

```
public static void ThrowIfGreaterThanOrEqual<T> (T value, T other, string?
paramName = default) where T : IComparable<T>;
```

Type Parameters

`T`

The type of the objects to validate.

Parameters

`value` T

The argument to validate as less than `other`.

`other` T

The value to compare with `value`.

`paramName` String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfLessThan<T>(T, T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is less than `other`.

```
C#

public static void ThrowIfLessThan<T> (T value, T other, string? paramName =
default) where T : IComparable<T>;
```

Type Parameters

T

The type of the objects to validate.

Parameters

value T

The argument to validate as greater than or equal to `other`.

other T

The value to compare with `value`.

paramName String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfLessThanOrEqual<T> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is less than or equal to `other`.

```
C#

public static void ThrowIfLessThanOrEqual<T> (T value, T other, string?
paramName = default) where T : IComparable<T>;
```

Type Parameters

T

The type of the objects to validate.

Parameters

value T

The argument to validate as greater than `other`.

other T

The value to compare with `value`.

paramName String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfNegative<T>(T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is negative.

```
C#

public static void ThrowIfNegative<T> (T value, string? paramName = default)
where T : System.Numerics.INumberBase<T>;
```

Type Parameters

T

The type of the object to validate.

Parameters

value T

The argument to validate as non-negative.

paramName String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfNegativeOrZero<T>(T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is negative or zero.

C#

```
public static void ThrowIfNegativeOrZero<T> (T value, string? paramName =
default) where T : System.Numerics.INumberBase<T>;
```

Type Parameters

T

The type of the object to validate.

Parameters

value T

The argument to validate as non-zero or non-negative.

paramName String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfNotEqual<T>(T, T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is not equal to `other`.

```
C#

public static void ThrowIfNotEqual<T> (T value, T other, string? paramName =
default) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the objects to validate.

Parameters

`value` T

The argument to validate as equal to `other`.

`other` T

The value to compare with `value`.

`paramName` String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArgumentOutOfRangeException.ThrowIfZero<T>(T, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws an ArgumentOutOfRangeException if `value` is zero.

```
C#

public static void ThrowIfZero<T> (T value, string? paramName = default)
where T : System.Numerics.INumberBase<T>;
```

Type Parameters

T

The type of the object to validate.

Parameters

value T

The argument to validate as non-zero.

paramName String

The name of the parameter with which `value` corresponds.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArithmeticException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown for errors in an arithmetic, casting, or conversion operation.

```
C#
public class ArithmeticException : SystemException
```

Inheritance Object → Exception → SystemException → ArithmeticException

Derived System.DivideByZeroException
System.NotFiniteNumberException
System.OverflowException

Remarks

ArithmeticException is the base class for the following exceptions:

- DivideByZeroException, which is thrown in integer division when the divisor is 0. For example, attempting to divide 10 by 0 throws a DivideByZeroException exception.
- NotFiniteNumberException, which is thrown when an operation is performed on or returns Double.NaN, Double.NegativeInfinity, Double.PositiveInfinity, Single.NaN, Single.NegativeInfinity, Single.PositiveInfinity, and the programming language used does not support those values.
- OverflowException, which is thrown when the result of an operation is outside the bounds of the target data type. That is, it is less than a number's `MinValue` property or greater than its `MaxValue` property. For example, attempting to assign 200 + 200 to a Byte value throws an OverflowException exception, since 400 greater than 256, the upper bound of the Byte data type.

Your code should not handle or throw this exception. Instead, you should either handle or throw one of its derived classes, since it more precisely indicates the exact nature of the error.

For a list of initial property values for an instance of ArithmeticException, see the ArithmeticException constructors.

ArithmeticException uses the HRESULT COR_E_ARITHMETIC, which has the value 0x80070216.

Constructors

Expand table

ArithmeticException()	Initializes a new instance of the ArithmeticException class.
ArithmeticException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArithmeticException class with serialized data.
ArithmeticException(String)	Initializes a new instance of the ArithmeticException class with a specified error message.
ArithmeticException(String, Exception)	Initializes a new instance of the ArithmeticException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)

Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArithmeticException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArithmeticException class.

Overloads

Expand table

ArithmeticException()	Initializes a new instance of the ArithmeticException class.
ArithmeticException(String)	Initializes a new instance of the ArithmeticException class with a specified error message.
ArithmeticException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArithmeticException class with serialized data.
ArithmeticException(String, Exception)	Initializes a new instance of the ArithmeticException class with a specified error message and a reference to the inner exception that is the cause of this exception.

ArithmeticException()

Initializes a new instance of the ArithmeticException class.

C#

```
public ArithmeticException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The arithmetic operation is not allowed." This message takes into account the current system culture.

The following table shows initial property values for an instance of ArithmeticException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArithmeticException(String)

Initializes a new instance of the ArithmeticException class with a specified error message.

C#

```
public ArithmeticException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows initial property values for an instance of ArithmeticException.

Expand table

Property	Value
InnerException	The null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArithmeticException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ArithmeticException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ArithmeticException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ArithmeticException(String, Exception)

Initializes a new instance of the ArithmeticException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public ArithmeticException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows initial property values for an instance of ArithmeticException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime.

C#

```
public abstract class Array : ICloneable, System.Collections.IList,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable
```

Inheritance Object → Array

Implements ICollection , IEnumerable , IList , IStructuralComparable , IStructuralEquatable , ICloneable

Examples

The following code example shows how Array.Copy copies elements between an array of type integer and an array of type Object.

C#

```
using System;
public class SamplesArray
{

    public static void Main()
    {

        // Creates and initializes a new integer array and a new Object
array.
        int[] myIntArray = new int[5] { 1, 2, 3, 4, 5 };
        Object[] myObjArray = new Object[5] { 26, 27, 28, 29, 30 };

        // Prints the initial values of both arrays.
        Console.WriteLine("Initially,");
```

```
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);

        // Copies the first two elements from the integer array to the
Object array.
        System.Array.Copy(myIntArray, myObjArray, 2);

        // Prints the values of the modified arrays.
        Console.WriteLine("\nAfter copying the first two elements of the
integer array to the Object array,");
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);

        // Copies the last two elements from the Object array to the integer
array.
        System.Array.Copy(myObjArray, myObjArray.GetUpperBound(0) - 1,
myIntArray, myIntArray.GetUpperBound(0) - 1, 2);

        // Prints the values of the modified arrays.
        Console.WriteLine("\nAfter copying the last two elements of the
Object array to the integer array,");
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);
    }

    public static void PrintValues(Object[] myArr)
    {
        foreach (Object i in myArr)
        {
            Console.Write("\t{0}", i);
        }
        Console.WriteLine();
    }

    public static void PrintValues(int[] myArr)
    {
        foreach (int i in myArr)
        {
            Console.Write("\t{0}", i);
        }
        Console.WriteLine();
    }
}
/*
This code produces the following output.

Initially,
integer array:  1       2       3       4       5
Object array:   26      27      28      29      30
```

```
After copying the first two elements of the integer array to the Object
array,
integer array:  1       2       3       4       5
Object array:   1       2       28      29      30

After copying the last two elements of the Object array to the integer
array,
integer array:  1       2       3       29      30
Object array:   1       2       28      29      30
*/
```

The following code example creates and initializes an Array and displays its properties and its elements.

C#

```
// Creates and initializes a new three-dimensional Array of type int.
Array myArr = Array.CreateInstance(typeof(int), 2, 3, 4);
for (int i = myArr.GetLowerBound(0); i <= myArr.GetUpperBound(0); i++)
{
    for (int j = myArr.GetLowerBound(1); j <= myArr.GetUpperBound(1); j++)
    {
        for (int k = myArr.GetLowerBound(2); k <= myArr.GetUpperBound(2);
k++)
        {
            myArr.SetValue((i * 100) + (j * 10) + k, i, j, k);
        }
    }
}

// Displays the properties of the Array.
Console.WriteLine("The Array has {0} dimension(s) and a total of {1}
elements.", myArr.Rank, myArr.Length);
Console.WriteLine("\tLength\tLower\tUpper");
for (int i = 0; i < myArr.Rank; i++)
{
    Console.Write("{0}:\t{1}", i, myArr.GetLength(i));
    Console.WriteLine("\t{0}\t{1}", myArr.GetLowerBound(i),
myArr.GetUpperBound(i));
}

// Displays the contents of the Array.
Console.WriteLine("The Array contains the following values:");
PrintValues(myArr);

void PrintValues(Array myArray)
{
    System.Collections.IEnumerator myEnumerator = myArray.GetEnumerator();
    int i = 0;
    int cols = myArray.GetLength(myArray.Rank - 1);
    while (myEnumerator.MoveNext())
    {
```

```
        if (i < cols)
        {
            i++;
        }
        else
        {
            Console.WriteLine();
            i = 1;
        }
        Console.Write("\t{0}", myEnumerator.Current);
    }
    Console.WriteLine();
}
// This code produces the following output.

// The Array has 3 dimension(s) and a total of 24 elements.
//     Length    Lower    Upper
// 0:  2    0    1
// 1:  3    0    2
// 2:  4    0    3
//
// The Array contains the following values:
//    0      1      2      3
//    10     11     12     13
//    20     21     22     23
//    100    101    102    103
//    110    111    112    113
//    120    121    122    123
```

Remarks

The Array class is not part of the System.Collections namespaces. However, it is still considered a collection because it is based on the IList interface.

The Array class is the base class for language implementations that support arrays. However, only the system and compilers can derive explicitly from the Array class. Users should employ the array constructs provided by the language.

An element is a value in an Array. The length of an Array is the total number of elements it can contain. The lower bound of an Array is the index of its first element. An Array can have any lower bound, but it has a lower bound of zero by default. A different lower bound can be defined when creating an instance of the Array class using CreateInstance. A multidimensional Array can have different bounds for each dimension. An array can have a maximum of 32 dimensions.

Unlike the classes in the System.Collections namespaces, Array has a fixed capacity. To increase the capacity, you must create a new Array object with the required capacity, copy the elements from the old Array object to the new one, and delete the old Array.

The array size is limited to a total of 4 billion elements, and to a maximum index of 0X7FEFFFFF in any given dimension (0X7FFFFFC7 for byte arrays and arrays of single-byte structures).

.NET Framework only: By default, the maximum size of an Array is 2 gigabytes (GB). In a 64-bit environment, you can avoid the size restriction by setting the `enabled` attribute of the gcAllowVeryLargeObjects configuration element to `true` in the run-time environment.

Single-dimensional arrays implement the System.Collections.Generic.IList<T>, System.Collections.Generic.ICollection<T>, System.Collections.Generic.IEnumerable<T>, System.Collections.Generic.IReadOnlyList<T> and System.Collections.Generic.IReadOnlyCollection<T> generic interfaces. The implementations are provided to arrays at run time, and as a result, the generic interfaces do not appear in the declaration syntax for the Array class. In addition, there are no reference topics for interface members that are accessible only by casting an array to the generic interface type (explicit interface implementations). The key thing to be aware of when you cast an array to one of these interfaces is that members which add, insert, or remove elements throw NotSupportedException.

Type objects provide information about array type declarations. Array objects with the same array type share the same Type object.

Type.IsArray and Type.GetElementType might not return the expected results with Array because if an array is cast to the type Array, the result is an object, not an array. That is, `typeof(System.Array).IsArray` returns `false`, and `typeof(System.Array).GetElementType` returns `null`.

The Array.Copy method copies elements not only between arrays of the same type but also between standard arrays of different types; it handles type casting automatically.

Some methods, such as CreateInstance, Copy, CopyTo, GetValue, and SetValue, provide overloads that accept 64-bit integers as parameters to accommodate large capacity arrays. LongLength and GetLongLength return 64-bit integers indicating the length of the array.

The Array is not guaranteed to be sorted. You must sort the Array prior to performing operations (such as BinarySearch) that require the Array to be sorted.

Using an Array object of pointers in native code is not supported and will throw a NotSupportedException for several methods.

Properties

Expand table

IsFixedSize	Gets a value indicating whether the Array has a fixed size.
IsReadOnly	Gets a value indicating whether the Array is read-only.
Is Synchronized	Gets a value indicating whether access to the Array is synchronized (thread safe).
Length	Gets the total number of elements in all the dimensions of the Array.
LongLength	Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
MaxLength	Gets the maximum number of elements that may be contained in an array.
Rank	Gets the rank (number of dimensions) of the Array. For example, a one-dimensional array returns 1, a two-dimensional array returns 2, and so on.
SyncRoot	Gets an object that can be used to synchronize access to the Array.

Methods

Expand table

AsReadOnly<T>(T[])	Returns a read-only wrapper for the specified array.
BinarySearch(Array, Int32, Int32, Object)	Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable interface implemented by each element of the array and by the specified value.
BinarySearch(Array, Int32, Int32, Object, IComparer)	Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer interface.
BinarySearch(Array, Object)	Searches an entire one-dimensional sorted array for a specific element, using the IComparable interface implemented by each element of the array and by the specified object.
BinarySearch(Array, Object, IComparer)	Searches an entire one-dimensional sorted array for a value using the specified IComparer interface.
BinarySearch<T>(T[], Int32, Int32, T)	Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable<T> generic

	interface implemented by each element of the Array and by the specified value.
BinarySearch<T>(T[], Int32, Int32, T, IComparer<T>)	Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer<T> generic interface.
BinarySearch<T>(T[], T)	Searches an entire one-dimensional sorted array for a specific element, using the IComparable<T> generic interface implemented by each element of the Array and by the specified object.
BinarySearch<T>(T[], T, IComparer<T>)	Searches an entire one-dimensional sorted array for a value using the specified IComparer<T> generic interface.
Clear(Array)	Clears the contents of an array.
Clear(Array, Int32, Int32)	Sets a range of elements in an array to the default value of each element type.
Clone()	Creates a shallow copy of the Array.
ConstrainedCopy(Array, Int32, Array, Int32, Int32)	Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. Guarantees that all changes are undone if the copy does not succeed completely.
ConvertAll<TInput,TOutput>(TInput[], Converter<TInput,TOutput>)	Converts an array of one type to an array of another type.
Copy(Array, Array, Int32)	Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
Copy(Array, Array, Int64)	Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 64-bit integer.
Copy(Array, Int32, Array, Int32, Int32)	Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 32-bit integers.
Copy(Array, Int64, Array, Int64, Int64)	Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 64-bit integers.

CopyTo(Array, Int32)	Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 32-bit integer.
CopyTo(Array, Int64)	Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 64-bit integer.
CreateInstance(Type, Int32)	Creates a one-dimensional Array of the specified Type and length, with zero-based indexing.
CreateInstance(Type, Int32, Int32)	Creates a two-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.
CreateInstance(Type, Int32, Int32, Int32)	Creates a three-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.
CreateInstance(Type, Int32[])	Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 32-bit integers.
CreateInstance(Type, Int32[], Int32[])	Creates a multidimensional Array of the specified Type and dimension lengths, with the specified lower bounds.
CreateInstance(Type, Int64[])	Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 64-bit integers.
Empty<T>()	Returns an empty array.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Exists<T>(T[], Predicate<T>)	Determines whether the specified array contains elements that match the conditions defined by the specified predicate.
Fill<T>(T[], T)	Assigns the given `value` of type `T` to each element of the specified `array`.
Fill<T>(T[], T, Int32, Int32)	Assigns the given `value` of type `T` to the elements of the specified `array` which are within the range of `startIndex` (inclusive) and the next `count` number of indices.
Find<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the first

	occurrence within the entire Array.
FindAll<T>(T[], Predicate<T>)	Retrieves all the elements that match the conditions defined by the specified predicate.
FindIndex<T>(T[], Int32, Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that starts at the specified index and contains the specified number of elements.
FindIndex<T>(T[], Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that extends from the specified index to the last element.
FindIndex<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the entire Array.
FindLast<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the last occurrence within the entire Array.
FindLastIndex<T>(T[], Int32, Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.
FindLastIndex<T>(T[], Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.
FindLastIndex<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the entire Array.
ForEach<T>(T[], Action<T>)	Performs the specified action on each element of the specified array.
GetEnumerator()	Returns an IEnumerator for the Array.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetLength(Int32)	Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.

GetLongLength(Int32)	Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
GetLowerBound(Int32)	Gets the index of the first element of the specified dimension in the array.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetUpperBound(Int32)	Gets the index of the last element of the specified dimension in the array.
GetValue(Int32)	Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
GetValue(Int32, Int32)	Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.
GetValue(Int32, Int32, Int32)	Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.
GetValue(Int32[])	Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.
GetValue(Int64)	Gets the value at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.
GetValue(Int64, Int64)	Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.
GetValue(Int64, Int64, Int64)	Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.
GetValue(Int64[])	Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.
IndexOf(Array, Object)	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.
IndexOf(Array, Object, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified index to the end of the array.

IndexOf(Array, Object, Int32, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of ifs first occurrence. The range extends from a specified index for a specified number of elements.
IndexOf<T>(T[], T)	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.
IndexOf<T>(T[], T, Int32)	Searches for the specified object in a range of elements of a one dimensional array, and returns the index of its first occurrence. The range extends from a specified index to the end of the array.
IndexOf<T>(T[], T, Int32, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified index for a specified number of elements.
Initialize()	Initializes every element of the value-type Array by calling the parameterless constructor of the value type.
LastIndexOf(Array, Object)	Searches for the specified object and returns the index of the last occurrence within the entire one-dimensional Array.
LastIndexOf(Array, Object, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that extends from the first element to the specified index.
LastIndexOf(Array, Object, Int32, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that contains the specified number of elements and ends at the specified index.
LastIndexOf<T>(T[], T)	Searches for the specified object and returns the index of the last occurrence within the entire Array.
LastIndexOf<T>(T[], T, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.
LastIndexOf<T>(T[], T, Int32, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)

Resize<T>(T[], Int32)	Changes the number of elements of a one-dimensional array to the specified new size.
Reverse(Array)	Reverses the sequence of the elements in the entire one-dimensional Array.
Reverse(Array, Int32, Int32)	Reverses the sequence of a subset of the elements in the one-dimensional Array.
Reverse<T>(T[])	Reverses the sequence of the elements in the one-dimensional generic array.
Reverse<T>(T[], Int32, Int32)	Reverses the sequence of a subset of the elements in the one-dimensional generic array.
SetValue(Object, Int32)	Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
SetValue(Object, Int32, Int32)	Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.
SetValue(Object, Int32, Int32, Int32)	Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.
SetValue(Object, Int32[])	Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.
SetValue(Object, Int64)	Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.
SetValue(Object, Int64, Int64)	Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.
SetValue(Object, Int64, Int64, Int64)	Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.
SetValue(Object, Int64[])	Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.
Sort(Array)	Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.

Sort(Array, Array)	Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.
Sort(Array, Array, IComparer)	Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.
Sort(Array, Array, Int32, Int32)	Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.
Sort(Array, Array, Int32, Int32, IComparer)	Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.
Sort(Array, IComparer)	Sorts the elements in a one-dimensional Array using the specified IComparer.
Sort(Array, Int32, Int32)	Sorts the elements in a range of elements in a one-dimensional Array using the IComparable implementation of each element of the Array.
Sort(Array, Int32, Int32, IComparer)	Sorts the elements in a range of elements in a one-dimensional Array using the specified IComparer.
Sort<T>(T[])	Sorts the elements in an entire Array using the IComparable<T> generic interface implementation of each element of the Array.
Sort<T>(T[], Comparison<T>)	Sorts the elements in an Array using the specified Comparison<T>.
Sort<T>(T[], IComparer<T>)	Sorts the elements in an Array using the specified IComparer<T> generic interface.
Sort<T>(T[], Int32, Int32)	Sorts the elements in a range of elements in an Array using the IComparable<T> generic interface implementation of each element of the Array.
Sort<T>(T[], Int32, Int32, IComparer<T>)	Sorts the elements in a range of elements in an Array using the specified IComparer<T> generic interface.
Sort<TKey,TValue>(TKey[], TValue[])	Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the

	keys in the first Array using the IComparable<T> generic interface implementation of each key.
Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>)	Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.
Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32)	Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable<T> generic interface implementation of each key.
Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>)	Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.
ToString()	Returns a string that represents the current object. (Inherited from Object)
TrueForAll<T>(T[], Predicate<T>)	Determines whether every element in the array matches the conditions defined by the specified predicate.

Explicit Interface Implementations

Expand table

ICollection.Count	Gets the number of elements contained in the Array.
IList.Add(Object)	Calling this method always throws a NotSupportedException exception.
IList.Clear()	Removes all items from the IList.
IList.Contains(Object)	Determines whether an element is in the IList.
IList.IndexOf(Object)	Determines the index of a specific item in the IList.
IList.Insert(Int32, Object)	Inserts an item to the IList at the specified index.
IList.Item[Int32]	Gets or sets the element at the specified index.
IList.Remove(Object)	Removes the first occurrence of a specific object from the IList.
IList.RemoveAt(Int32)	Removes the IList item at the specified index.

IStructuralComparable.CompareTo(Object, IComparer)	Determines whether the current collection object precedes, occurs in the same position as, or follows another object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Determines whether an object is equal to the current instance.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Returns a hash code for the current instance.

Extension Methods

Expand table

Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

Public static (`Shared` in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

This implementation does not provide a synchronized (thread safe) wrapper for an Array; however, .NET classes based on Array provide their own synchronized version of the collection using the SyncRoot property.

Enumerating through a collection is intrinsically not a thread-safe procedure. Even when a collection is synchronized, other threads can still modify the collection, which causes the enumerator to throw an exception. To guarantee thread safety during enumeration, you can either lock the collection during the entire enumeration or catch the exceptions resulting from changes made by other threads.

See also

- Object
- Type
- Performing Culture-Insensitive String Operations in Arrays
- Arrays (C# Programming Guide)
- Arrays in F#
- Arrays in Visual Basic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IsFixedSize Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Array has a fixed size.

```
C#
public bool IsFixedSize { get; }
```

Property Value

Boolean

This property is always `true` for all arrays.

Implements

IsFixedSize

Remarks

Array implements the IsFixedSize property because it is required by the System.Collections.IList interface.

An array with a fixed size does not allow the addition or removal of elements after the array is created, but it allows the modification of existing elements.

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IList
- IList<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IsReadOnly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Array is read-only.

C#

```
public bool IsReadOnly { get; }
```

Property Value

Boolean

This property is always `false` for all arrays.

Implements

IsReadOnly

Remarks

Array implements the IsReadOnly property because it is required by the System.Collections.IList interface. An array that is read-only does not allow the addition, removal, or modification of elements after the array is created.

If you require a read-only collection, use a System.Collections class that implements the System.Collections.IList interface.

If you cast or convert an array to an IList interface object, the IList.IsReadOnly property returns `false`. However, if you cast or convert an array to a IList<T> interface, the `IsReadOnly` property returns `true`.

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IList
- System.Collections
- IList<T>
- System.Collections.Generic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IsSynchronized Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether access to the Array is synchronized (thread safe).

C#

```
public bool IsSynchronized { get; }
```

Property Value

Boolean

This property is always `false` for all arrays.

Implements

IsSynchronized

Examples

The following code example shows how to lock an array during the entire enumeration by using the SyncRoot property.

C#

```
Array myArray = new int[] { 1, 2, 4 };
lock(myArray.SyncRoot)
{
    foreach (Object item in myArray)
        Console.WriteLine(item);
}
```

Remarks

Array implements the IsSynchronized property because it is required by the System.Collections.ICollection interface.

.NET classes based on Array provide their own synchronized version of the collection using the SyncRoot property.

Classes that use arrays can also implement their own synchronization using the SyncRoot property. The synchronizing code must perform operations on the `SyncRoot` of the collection, not directly on the collection. This ensures proper operation of collections that are derived from other objects. Specifically, it maintains proper synchronization with other threads that might be simultaneously modifying the collection. Note that some implementations of SyncRoot might return the Array itself.

Enumerating through a collection is intrinsically not a thread safe procedure. Even when a collection is synchronized, other threads can still modify the collection, which causes the enumerator to throw an exception. To guarantee thread safety during enumeration, you can either lock the collection during the entire enumeration or catch the exceptions resulting from changes made by other threads.

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SyncRoot
- ICollection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Array.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total number of elements in all the dimensions of the Array.

C#

```
public int Length { get; }
```

Property Value

Int32

The total number of elements in all the dimensions of the Array; zero if there are no elements in the array.

Exceptions

OverflowException

The array is multidimensional and contains more than Int32.MaxValue elements.

Examples

The following example uses the Length property to get the total number of elements in an array. It also uses the GetUpperBound method to determine the number of elements in each dimension of a multidimensional array.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        // Declare a single-dimensional string array
        String[] array1d = { "zero", "one", "two", "three" };
```

```
        ShowArrayInfo(array1d);

        // Declare a two-dimensional string array
        String[,] array2d = { { "zero", "0" }, { "one", "1" },
                              { "two", "2" }, { "three", "3"},
                              { "four", "4" }, { "five", "5" } };
        ShowArrayInfo(array2d);

        // Declare a three-dimensional integer array
        int[, ,] array3d = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                         { { 7, 8, 9 }, { 10, 11, 12 } } };
        ShowArrayInfo(array3d);
    }

    private static void ShowArrayInfo(Array arr)
    {
        Console.WriteLine("Length of Array:      {0,3}", arr.Length);
        Console.WriteLine("Number of Dimensions: {0,3}", arr.Rank);
        // For multidimensional arrays, show number of elements in each
dimension.
        if (arr.Rank > 1) {
            for (int dimension = 1; dimension <= arr.Rank; dimension++)
                Console.WriteLine("   Dimension {0}: {1,3}", dimension,
                                  arr.GetUpperBound(dimension - 1) + 1);
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//       Length of Array:        4
//       Number of Dimensions:   1
//
//       Length of Array:       12
//       Number of Dimensions:   2
//          Dimension 1:   6
//          Dimension 2:   2
//
//       Length of Array:       12
//       Number of Dimensions:   3
//          Dimension 1:   2
//          Dimension 2:   2
//          Dimension 3:   3
```

Remarks

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetLength(Int32)
- GetUpperBound(Int32)
- GetLowerBound(Int32)
- LongLength
- Count
- Rank

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.LongLength Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.

C#

```
public long LongLength { get; }
```

Property Value

Int64

A 64-bit integer that represents the total number of elements in all the dimensions of the Array.

Remarks

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetLongLength(Int32)
- Length

- Count
- Rank

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.MaxLength Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum number of elements that may be contained in an array.

C#

```
public static int MaxLength { get; }
```

Property Value

Int32

The maximum count of elements allowed in any array.

Remarks

This property represents a runtime limitation, the maximum number of elements (not bytes) the runtime will allow in an array. There is no guarantee that an allocation under this length will succeed, but all attempts to allocate a larger array will fail.

This property only applies to single-dimension, zero-bound (SZ) arrays. Length property may return larger value than this property for multi-dimensional arrays.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.Rank Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the rank (number of dimensions) of the Array. For example, a one-dimensional array returns 1, a two-dimensional array returns 2, and so on.

C#

```
public int Rank { get; }
```

Property Value

Int32

The rank (number of dimensions) of the Array.

Examples

The following example initializes a one-dimensional array, a two-dimensional array, and a jagged array, and retrieves the Rank property of each.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int[] array1 = new int[10];
      int[,] array2= new int[10,3];
      int[][] array3 = new int[10][];

      Console.WriteLine("{0}: {1} dimension(s)",
                        array1.ToString(), array1.Rank);
      Console.WriteLine("{0}: {1} dimension(s)",
                        array2.ToString(), array2.Rank);
      Console.WriteLine("{0}: {1} dimension(s)",
                        array3.ToString(), array3.Rank);
```

```
    }
}
// The example displays the following output:
//       System.Int32[]: 1 dimension(s)
//       System.Int32[,]: 2 dimension(s)
//       System.Int32[][]: 1 dimension(s)
```

Remarks

For example, the following code creates an array of three dimensions with a Rank property whose value is 3.

C#

```
int[,,] TDArray = new int[1,1,1];
```

A jagged array (an array of arrays) is a one-dimensional array; the value of its Rank property is 1.

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Length
- GetLength(Int32)
- LongLength
- GetLongLength(Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.SyncRoot Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an object that can be used to synchronize access to the Array.

C#

```
public object SyncRoot { get; }
```

Property Value

Object

An object that can be used to synchronize access to the Array.

Implements

SyncRoot

Examples

The following code example shows how to lock an array during the entire enumeration by using the SyncRoot property.

C#

```
Array myArray = new int[] { 1, 2, 4 };
lock(myArray.SyncRoot)
{
    foreach (Object item in myArray)
        Console.WriteLine(item);
}
```

Remarks

This property implements the System.Collections.ICollection interface.

.NET classes based on Array provide their own synchronized version of the collection using the SyncRoot property.

Classes that use arrays can also implement their own synchronization using the SyncRoot property. The synchronizing code must perform operations on the `SyncRoot` of the collection, not directly on the collection. This ensures proper operation of collections that are derived from other objects. Specifically, it maintains proper synchronization with other threads that might be simultaneously modifying the collection. Note that some implementations of SyncRoot might return the Array itself.

Enumerating through a collection is intrinsically not a thread safe procedure. Even when a collection is synchronized, other threads can still modify the collection, which causes the enumerator to throw an exception. To guarantee thread safety during enumeration, you can either lock the collection during the entire enumeration or catch the exceptions resulting from changes made by other threads.

Retrieving the value of this property is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsSynchronized

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Array.AsReadOnly<T>(T[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a read-only wrapper for the specified array.

C#

```
public static System.Collections.ObjectModel.ReadOnlyCollection<T>
AsReadOnly<T> (T[] array);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based array to wrap in a read-only ReadOnlyCollection<T> wrapper.

Returns

ReadOnlyCollection<T>

A read-only ReadOnlyCollection<T> wrapper for the specified array.

Exceptions

ArgumentNullException

`array` is `null`.

Examples

The following example wraps an array in a read-only ReadOnlyCollection<T>.

C#

```
using System;
using System.Collections.Generic;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
      String[] myArr = { "The", "quick", "brown", "fox" };

      // Display the values of the array.
      Console.WriteLine( "The string array initially contains the following
values:" );
      PrintIndexAndValues( myArr );

      // Create a read-only IList wrapper around the array.
      IList<string> myList = Array.AsReadOnly( myArr );

      // Display the values of the read-only IList.
      Console.WriteLine( "The read-only IList contains the following
values:" );
      PrintIndexAndValues( myList );

      // Attempt to change a value through the wrapper.
      try  {
         myList[3] = "CAT";
      }
      catch ( NotSupportedException e )  {
         Console.WriteLine( "{0} - {1}", e.GetType(), e.Message );
         Console.WriteLine();
      }

      // Change a value in the original array.
      myArr[2] = "RED";

      // Display the values of the array.
      Console.WriteLine( "After changing the third element, the string array
contains the following values:" );
      PrintIndexAndValues( myArr );

      // Display the values of the read-only IList.
      Console.WriteLine( "After changing the third element, the read-only
IList contains the following values:" );
      PrintIndexAndValues( myList );
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
```

```
    public static void PrintIndexAndValues( IList<string> myList )  {
        for ( int i = 0; i < myList.Count; i++ )  {
            Console.WriteLine( "   [{0}] : {1}", i, myList[i] );
        }
        Console.WriteLine();
    }
}


/*
This code produces the following output.

The string array initially contains the following values:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox

The read-only IList contains the following values:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox

System.NotSupportedException - Collection is read-only.

After changing the third element, the string array contains the following
values:
   [0] : The
   [1] : quick
   [2] : RED
   [3] : fox

After changing the third element, the read-only IList contains the following
values:
   [0] : The
   [1] : quick
   [2] : RED
   [3] : fox

*/
```

Remarks

To prevent any modifications to the array, expose the array only through this wrapper.

A collection that is read-only is simply a collection with a wrapper that prevents modifying the collection; therefore, if changes are made to the underlying collection, the read-only collection reflects those changes.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.BinarySearch Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches a one-dimensional sorted Array for a value, using a binary search algorithm.

Overloads

Expand table

BinarySearch(Array, Object)	Searches an entire one-dimensional sorted array for a specific element, using the IComparable interface implemented by each element of the array and by the specified object.
BinarySearch(Array, Object, IComparer)	Searches an entire one-dimensional sorted array for a value using the specified IComparer interface.
BinarySearch(Array, Int32, Int32, Object)	Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable interface implemented by each element of the array and by the specified value.
BinarySearch(Array, Int32, Int32, Object, IComparer)	Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer interface.
BinarySearch<T>(T[], T)	Searches an entire one-dimensional sorted array for a specific element, using the IComparable<T> generic interface implemented by each element of the Array and by the specified object.
BinarySearch<T>(T[], T, IComparer<T>)	Searches an entire one-dimensional sorted array for a value using the specified IComparer<T> generic interface.
BinarySearch<T>(T[], Int32, Int32, T)	Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable<T> generic interface implemented by each element of the Array and by the specified value.
BinarySearch<T>(T[], Int32, Int32, T, IComparer<T>)	Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer<T> generic interface.

BinarySearch(Array, Object)

Searches an entire one-dimensional sorted array for a specific element, using the IComparable interface implemented by each element of the array and by the specified object.

```
C#
public static int BinarySearch (Array array, object? value);
```

Parameters

`array` Array

The sorted one-dimensional Array to search.

`value` Object

The object to search for.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentException

`value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`value` does not implement the IComparable interface, and the search encounters an element that does not implement the IComparable interface.

Examples

The following code example shows how to use BinarySearch to locate a specific object in an Array.

> **Note**
>
> The array is created with its elements in ascending sort order. The **BinarySearch** method requires the array to be sorted in ascending order.

C#

```
using System;

public class SamplesArray
{
    public static void Main()
    {
        // Creates and initializes a new Array.
        Array myIntArray = Array.CreateInstance(typeof(int), 5);

        myIntArray.SetValue(8, 0);
        myIntArray.SetValue(2, 1);
        myIntArray.SetValue(6, 2);
        myIntArray.SetValue(3, 3);
        myIntArray.SetValue(7, 4);

        // Do the required sort first
        Array.Sort(myIntArray);

        // Displays the values of the Array.
        Console.WriteLine( "The int array contains the following:" );
        PrintValues(myIntArray);

        // Locates a specific object that does not exist in the Array.
        object myObjectOdd = 1;
        FindMyObject( myIntArray, myObjectOdd );

        // Locates an object that exists in the Array.
        object myObjectEven = 6;
        FindMyObject(myIntArray, myObjectEven);
    }
```

```
    public static void FindMyObject(Array myArr, object myObject)
    {
        int myIndex=Array.BinarySearch(myArr, myObject);
        if (myIndex < 0)
        {
            Console.WriteLine("The object to search for ({0}) is not
found. The next larger object is at index {1}.", myObject, ~myIndex );
        }
        else
        {
            Console.WriteLine("The object to search for ({0}) is at index
{1}.", myObject, myIndex );
        }
    }

    public static void PrintValues(Array myArr)
    {
        int i = 0;
        int cols = myArr.GetLength(myArr.Rank - 1);
        foreach (object o in myArr)
        {
            if ( i < cols )
            {
                i++;
            }
            else
            {
                Console.WriteLine();
                i = 1;
            }
            Console.Write( "\t{0}", o);
        }
        Console.WriteLine();
    }
}
// This code produces the following output.
//
//The int array contains the following:
//        2       3       6       7       8
//The object to search for (1) is not found. The next larger object is at
index 0
//
//The object to search for (6) is at index 2.
```

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the Array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is one greater than the upper bound of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

Either `value` or every element of `array` must implement the IComparable interface, which is used for comparisons. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable implementation; otherwise, the result might be incorrect.

> ⓘ **Note**
>
> If `value` does not implement the **IComparable** interface, the elements of `array` are not tested for **IComparable** before the search begins. An exception is thrown if the search encounters an element that does not implement **IComparable**.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> ⓘ **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable** implementation, even if `value` is `null`. That is, the **IComparable** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is the Length of `array`.

See also

- IComparable
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch(Array, Object, IComparer)

Searches an entire one-dimensional sorted array for a value using the specified IComparer interface.

C#

```
public static int BinarySearch (Array array, object? value,
System.Collections.IComparer? comparer);
```

Parameters

`array` Array

The sorted one-dimensional Array to search.

`value` Object

The object to search for.

`comparer` IComparer

The IComparer implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentException

`comparer` is `null`, and `value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`comparer` is `null`, `value` does not implement the IComparable interface, and the search encounters an element that does not implement the IComparable interface.

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the Array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is one greater than the upper bound of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

The comparer customizes how the elements are compared. For example, you can use a System.Collections.CaseInsensitiveComparer as the comparer to perform case-

insensitive string searches.

If `comparer` is not `null`, the elements of `array` are compared to the specified value using the specified IComparer implementation. The elements of `array` must already be sorted in increasing value according to the sort order defined by `comparer`; otherwise, the result might be incorrect.

If `comparer` is `null`, the comparison is done using the IComparable implementation provided by the element itself or by the specified value. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable implementation; otherwise, the result might be incorrect.

> ⓘ **Note**
>
> If `comparer` is `null` and `value` does not implement the **IComparable** interface, the elements of `array` are not tested for **IComparable** before the search begins. An exception is thrown if the search encounters an element that does not implement **IComparable**.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> ⓘ **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable** implementation, even if `value` is `null`. That is, the **IComparable** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is the Length of `array`.

See also

- IComparer
- IComparable
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch(Array, Int32, Int32, Object)

Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable interface implemented by each element of the array and by the specified value.

C#

```
public static int BinarySearch (Array array, int index, int length,
object? value);
```

Parameters

`array` Array

The sorted one-dimensional Array to search.

`index` Int32

The starting index of the range to search.

`length` Int32

The length of the range to search.

`value` Object

The object to search for.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

`value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`value` does not implement the IComparable interface, and the search encounters an element that does not implement the IComparable interface.

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the Array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is one greater than the upper bound of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

Either `value` or every element of `array` must implement the IComparable interface, which is used for comparisons. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable implementation; otherwise, the result might be incorrect.

> ⓘ **Note**
>
> If `value` does not implement the **IComparable** interface, the elements of `array` are not tested for **IComparable** before the search begins. An exception is thrown if the search encounters an element that does not implement **IComparable**.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> ⓘ **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable** implementation, even if `value` is `null`. That is, the **IComparable** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is `length`.

See also

- IComparable
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch(Array, Int32, Int32, Object, IComparer)

Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer interface.

C#

```
public static int BinarySearch (Array array, int index, int length,
object? value, System.Collections.IComparer? comparer);
```

Parameters

`array` Array

The sorted one-dimensional Array to search.

`index` Int32

The starting index of the range to search.

`length` Int32

The length of the range to search.

`value` Object

The object to search for.

`comparer` IComparer

The IComparer implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

`comparer` is `null`, and `value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`comparer` is `null`, `value` does not implement the IComparable interface, and the search encounters an element that does not implement the IComparable interface.

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the Array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is one greater than the upper bound of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

The comparer customizes how the elements are compared. For example, you can use a System.Collections.CaseInsensitiveComparer as the comparer to perform case-insensitive string searches.

If `comparer` is not `null`, the elements of `array` are compared to the specified value using the specified IComparer implementation. The elements of `array` must already be sorted in increasing value according to the sort order defined by `comparer`; otherwise, the result might be incorrect.

If `comparer` is `null`, the comparison is done using the IComparable implementation provided by the element itself or by the specified value. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable implementation; otherwise, the result might be incorrect.

> **Note**
>
> If `comparer` is `null` and `value` does not implement the **IComparable** interface, the elements of `array` are not tested for **IComparable** before the search begins. An exception is thrown if the search encounters an element that does not implement **IComparable**.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception when using IComparable.

> **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable** implementation, even if `value` is `null`. That is, the **IComparable** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is `length`.

See also

- IComparer
- IComparable
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch<T>(T[], T)

Searches an entire one-dimensional sorted array for a specific element, using the IComparable<T> generic interface implemented by each element of the Array and by the specified object.

C#

```
public static int BinarySearch<T> (T[] array, T value);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The sorted one-dimensional, zero-based Array to search.

`value` T

The object to search for.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

InvalidOperationException

`T` does not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<T>(T[]) generic method overload and the BinarySearch<T>(T[], T) generic method overload. An array of strings is created, in no particular order.

The array is displayed, sorted, and displayed again. Arrays must be sorted in order to use the BinarySearch method.

> **Note**
>
> The calls to the **Sort** and **BinarySearch** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, F#, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

The BinarySearch<T>(T[], T) generic method overload is then used to search for two strings, one that is not in the array and one that is. The array and the return value of the BinarySearch method are passed to the `ShowWhere` generic method (the `showWhere` function in the F# example), which displays the index value if the string is found, and otherwise the elements the search string would fall between if it were in the array. The index is negative if the string is not in the array, so the `ShowWhere` method takes the bitwise complement (the ~ operator in C# and Visual C++, the ~~~ operator in F#, `Xor`-1 in Visual Basic) to obtain the index of the first element in the list that is larger than the search string.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Tyrannosaurus",
                              "Mamenchisaurus",
                              "Deinonychus",
                              "Edmontosaurus"};

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nSort");
        Array.Sort(dinosaurs);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
```

```
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nBinarySearch for 'Coelophysis':");
        int index = Array.BinarySearch(dinosaurs, "Coelophysis");
        ShowWhere(dinosaurs, index);

        Console.WriteLine("\nBinarySearch for 'Tyrannosaurus':");
        index = Array.BinarySearch(dinosaurs, "Tyrannosaurus");
        ShowWhere(dinosaurs, index);
    }

    private static void ShowWhere<T>(T[] array, int index)
    {
        if (index<0)
        {
            // If the index is negative, it represents the bitwise
            // complement of the next larger element in the array.
            //
            index = ~index;

            Console.Write("Not found. Sorts between: ");

            if (index == 0)
                Console.Write("beginning of array and ");
            else
                Console.Write("{0} and ", array[index-1]);

            if (index == array.Length)
                Console.WriteLine("end of array.");
            else
                Console.WriteLine("{0}.", array[index]);
        }
        else
        {
            Console.WriteLine("Found at index {0}.", index);
        }
    }
}

/* This code example produces the following output:

Pachycephalosaurus
Amargasaurus
Tyrannosaurus
Mamenchisaurus
Deinonychus
Edmontosaurus

Sort

Amargasaurus
Deinonychus
Edmontosaurus
Mamenchisaurus
```

```
Pachycephalosaurus
Tyrannosaurus

BinarySearch for 'Coelophysis':
Not found. Sorts between: Amargasaurus and Deinonychus.

BinarySearch for 'Tyrannosaurus':
Found at index 5.
 */
```

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If `array` does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is equal to the size of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

`T` must implement the IComparable<T> generic interface, which is used for comparisons. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable<T> implementation; otherwise, the result might be incorrect.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable<T>** implementation, even if `value` is `null`. That is, the **IComparable<T>** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is the Length of `array`.

See also

- IComparable<T>
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch<T>(T[], T, IComparer<T>)

Searches an entire one-dimensional sorted array for a value using the specified IComparer<T> generic interface.

C#

```
public static int BinarySearch<T> (T[] array, T value,
System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The sorted one-dimensional, zero-based Array to search.

`value` T

The object to search for.

`comparer` IComparer<T>

The IComparer<T> implementation to use when comparing elements.

-or-

`null` to use the IComparable<T> implementation of each element.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentException

`comparer` is `null`, and `value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`comparer` is `null`, and `T` does not implement the IComparable<T> generic interface

Examples

The following example demonstrates the Sort<T>(T[], IComparer<T>) generic method overload and the BinarySearch<T>(T[], T, IComparer<T>) generic method overload.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls

the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The array is displayed, sorted, and displayed again. Arrays must be sorted in order to use the BinarySearch method.

> ⓘ **Note**
>
> The calls to the **Sort<T>(T[], IComparer<T>)** and **BinarySearch<T>(T[], T, IComparer<T>)** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

The BinarySearch<T>(T[], T, IComparer<T>) generic method overload is then used to search for two strings, one that is not in the array and one that is. The array and the return value of the BinarySearch<T>(T[], T, IComparer<T>) method are passed to the `ShowWhere` generic method (the `showWhere` function in the F# example), which displays the index value if the string is found, and otherwise the elements the search string would fall between if it were in the array. The index is negative if the string is not n the array, so the `ShowWhere` method takes the bitwise complement (the ~ operator in C# and Visual C++, the ~~~ operator in F#, `Xor` -1 in Visual Basic) to obtain the index of the first element in the list that is larger than the search string.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Tyrannosaurus",
```

```
                                "Mamenchisaurus",
                                "Deinonychus",
                                "Edmontosaurus"};

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort");
        Array.Sort(dinosaurs, rc);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nBinarySearch for 'Coelophysis':");
        int index = Array.BinarySearch(dinosaurs, "Coelophysis", rc);
        ShowWhere(dinosaurs, index);

        Console.WriteLine("\nBinarySearch for 'Tyrannosaurus':");
        index = Array.BinarySearch(dinosaurs, "Tyrannosaurus", rc);
        ShowWhere(dinosaurs, index);
    }

    private static void ShowWhere<T>(T[] array, int index)
    {
        if (index<0)
        {
            // If the index is negative, it represents the bitwise
            // complement of the next larger element in the array.
            //
            index = ~index;

            Console.Write("Not found. Sorts between: ");

            if (index == 0)
                Console.Write("beginning of array and ");
            else
                Console.Write("{0} and ", array[index-1]);

            if (index == array.Length)
                Console.WriteLine("end of array.");
            else
                Console.WriteLine("{0}.", array[index]);
        }
        else
        {
            Console.WriteLine("Found at index {0}.", index);
        }
```

```
    }
}

/* This code example produces the following output:

Pachycephalosaurus
Amargasaurus
Tyrannosaurus
Mamenchisaurus
Deinonychus
Edmontosaurus

Sort

Tyrannosaurus
Pachycephalosaurus
Mamenchisaurus
Edmontosaurus
Deinonychus
Amargasaurus

BinarySearch for 'Coelophysis':
Not found. Sorts between: Deinonychus and Amargasaurus.

BinarySearch for 'Tyrannosaurus':
Found at index 0.
 */
```

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the Array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is equal to the size of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

The comparer customizes how the elements are compared. For example, you can use a System.Collections.CaseInsensitiveComparer as the comparer to perform case-insensitive string searches.

If `comparer` is not `null`, the elements of `array` are compared to the specified value using the specified IComparer<T> generic interface implementation. The elements of `array` must already be sorted in increasing value according to the sort order defined by `comparer`; otherwise, the result might be incorrect.

If `comparer` is `null`, the comparison is done using the IComparable<T> generic interface implementation provided by `T`. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable<T> implementation; otherwise, the result might be incorrect.

> **Note**
>
> If `comparer` is `null` and `value` does not implement the **IComparable<T>** generic interface, the elements of `array` are not tested for **IComparable<T>** before the search begins. An exception is thrown if the search encounters an element that does not implement **IComparable<T>**.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable<T>** implementation, even if `value` is `null`. That is, the **IComparable<T>** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is the Length of `array`.

See also

- IComparer<T>
- IComparable<T>
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch<T>(T[], Int32, Int32, T)

Searches a range of elements in a one-dimensional sorted array for a value, using the IComparable<T> generic interface implemented by each element of the Array and by the specified value.

C#

```
public static int BinarySearch<T> (T[] array, int index, int length, T
value);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The sorted one-dimensional, zero-based Array to search.

`index` Int32

The starting index of the range to search.

`length` Int32

The length of the range to search.

`value` T

The object to search for.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

`value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`T` does not implement the IComparable<T> generic interface.

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual

Basic) to the negative result to produce an index. If this index is equal to the size of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

`T` must implement the IComparable<T> generic interface, which is used for comparisons. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable<T> implementation; otherwise, the result might be incorrect.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception.

> ⓘ **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable<T>** implementation, even if `value` is `null`. That is, the **IComparable<T>** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is `length`.

See also

- IComparable<T>
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BinarySearch<T>(T[], Int32, Int32, T, IComparer<T>)

Searches a range of elements in a one-dimensional sorted array for a value, using the specified IComparer<T> generic interface.

C#

```
public static int BinarySearch<T> (T[] array, int index, int length, T
value, System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The sorted one-dimensional, zero-based Array to search.

`index` Int32

The starting index of the range to search.

`length` Int32

The length of the range to search.

`value` T

The object to search for.

`comparer` IComparer<T>

The IComparer<T> implementation to use when comparing elements.

-or-

`null` to use the IComparable<T> implementation of each element.

Returns

Int32

The index of the specified `value` in the specified `array`, if `value` is found; otherwise, a negative number. If `value` is not found and `value` is less than one or more elements in `array`, the negative number returned is the bitwise complement of the index of the first element that is larger than `value`. If `value` is not found and `value` is greater than all elements in `array`, the negative number returned is the bitwise complement of (the index of the last element plus 1). If this method is called with a non-sorted `array`, the return value can be incorrect and a negative number could be returned, even if `value` is present in `array`.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

`comparer` is `null`, and `value` is of a type that is not compatible with the elements of `array`.

InvalidOperationException

`comparer` is `null`, and `T` does not implement the IComparable<T> generic interface.

Remarks

This method does not support searching arrays that contain negative indexes. `array` must be sorted before calling this method.

If the array does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operator (~ in C#, `Not` in Visual Basic) to the negative result to produce an index. If this index is equal to the size of the array, there are no elements larger than `value` in the array. Otherwise, it is the index of the first element that is larger than `value`.

The comparer customizes how the elements are compared. For example, you can use a System.Collections.CaseInsensitiveComparer as the comparer to perform case-insensitive string searches.

If `comparer` is not `null`, the elements of `array` are compared to the specified value using the specified IComparer<T> generic interface implementation. The elements of `array` must already be sorted in increasing value according to the sort order defined by `comparer`; otherwise, the result might be incorrect.

If `comparer` is `null`, the comparison is done using the IComparable<T> generic interface implementation provided for type `T`. The elements of `array` must already be sorted in increasing value according to the sort order defined by the IComparable<T> implementation; otherwise, the result might be incorrect.

Duplicate elements are allowed. If the Array contains more than one element equal to `value`, the method returns the index of only one of the occurrences, and not necessarily the first one.

`null` can always be compared with any other reference type; therefore, comparisons with `null` do not generate an exception when using IComparable<T>.

> **Note**
>
> For every element tested, `value` is passed to the appropriate **IComparable<T>** implementation, even if `value` is `null`. That is, the **IComparable<T>** implementation determines how a given element compares to `null`.

This method is an O(log `n`) operation, where `n` is `length`.

See also

- IComparer<T>
- IComparable<T>
- Sort
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Clear Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Clear(Array)	Clears the contents of an array.
Clear(Array, Int32, Int32)	Sets a range of elements in an array to the default value of each element type.

Clear(Array)

Clears the contents of an array.

C#

```
public static void Clear (Array array);
```

Parameters

`array` Array

The array to clear.

Exceptions

ArgumentNullException

`array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Clear(Array, Int32, Int32)

Sets a range of elements in an array to the default value of each element type.

```
C#
public static void Clear (Array array, int index, int length);
```

Parameters

`array` Array

The array whose elements need to be cleared.

`index` Int32

The starting index of the range of elements to clear.

`length` Int32

The number of elements to clear.

Exceptions

ArgumentNullException

`array` is `null`.

IndexOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

-or-

The sum of `index` and `length` is greater than the size of `array`.

Examples

The following example uses the Clear method to reset integer values in a one-dimensional, two-dimensional, and three-dimensional array.

C#

```
using System;

class Example
{
    public static void Main()
    {
        Console.WriteLine("One dimension (Rank=1):");
        int[] numbers1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

        for (int i = 0; i < 9; i++)
        {
            Console.Write("{0} ", numbers1[i]);
        }
        Console.WriteLine();
        Console.WriteLine();

        Console.WriteLine("Array.Clear(numbers1, 2, 5)");
        Array.Clear(numbers1, 2, 5);

        for (int i = 0; i < 9; i++)
        {
            Console.Write("{0} ", numbers1[i]);
        }
        Console.WriteLine();
        Console.WriteLine();

        Console.WriteLine("Two dimensions (Rank=2):");
        int[,] numbers2 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                Console.Write("{0} ", numbers2[i, j]);
            }
            Console.WriteLine();
        }

        Console.WriteLine();
        Console.WriteLine("Array.Clear(numbers2, 2, 5)");
        Array.Clear(numbers2, 2, 5);

        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                Console.Write("{0} ", numbers2[i, j]);
            }
            Console.WriteLine();
```

```
        }

        Console.WriteLine("Three dimensions (Rank=3):");
        int[, ,] numbers3 = {{{1, 2}, {3, 4}},
                             {{5, 6}, {7, 8}},
                             {{9, 10}, {11, 12}}};

        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                for (int k = 0; k < 2; k++)
                {
                    Console.Write("{0} ", numbers3[i, j, k]);
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }

        Console.WriteLine("Array.Clear(numbers3, 2, 5)");
        Array.Clear(numbers3, 2, 5);

        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                for (int k = 0; k < 2; k++)
                {
                    Console.Write("{0} ", numbers3[i, j, k]);
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }
    }
}
/*  This code example produces the following output:
 *
 * One dimension (Rank=1):
 * 1 2 3 4 5 6 7 8 9
 *
 * Array.Clear(numbers1, 2, 5)
 * 1 2 0 0 0 0 0 8 9
 *
 * Two dimensions (Rank=2):
 * 1 2 3
 * 4 5 6
 * 7 8 9
 *
 * Array.Clear(numbers2, 2, 5)
 * 1 2 0
 * 0 0 0
 * 0 8 9
 *
```

```
 * Three dimensions (Rank=3):
 * 1 2
 * 3 4
 *
 * 5 6
 * 7 8
 *
 * Array.Clear(numbers3, 2, 5)
 * 1 2
 * 0 0
 *
 * 0 0
 * 0 8
 */
```

The following example defines a `TimeZoneTime` structure that includes a TimeZoneInfo field and a DateTimeOffset field. It then calls the Clear method to clear one element in a two-element array of `TimeZoneTime` values. The method sets the value of the cleared element to the default value of a TimeZoneInfo object, which is `null`, and the default value of a DateTimeOffset object, which is DateTimeOffset.MinValue.

C#

```
using System;

public struct TimeZoneTime
{
   private DateTimeOffset dt;
   private TimeZoneInfo tz;

   public TimeZoneTime(DateTimeOffset dateTime, TimeZoneInfo timeZone)
   {
      dt = dateTime;
      tz = timeZone;
   }

   public DateTimeOffset DateTime
   { get { return dt; } }

   public TimeZoneInfo TimeZone
   { get { return tz; } }
}

public class Example
{
   public static void Main()
   {
      // Declare an array with two elements.
      TimeZoneTime[] timeZoneTimes = { new TimeZoneTime(DateTime.Now,
TimeZoneInfo.Local),
```

```
                                        new TimeZoneTime(DateTime.Now,
TimeZoneInfo.Utc) };
      foreach (var timeZoneTime in timeZoneTimes)
         Console.WriteLine("{0}: {1:G}",
                           timeZoneTime.TimeZone == null ? "<null>" :
timeZoneTime.TimeZone.ToString(),
                           timeZoneTime.DateTime);
      Console.WriteLine();

      Array.Clear(timeZoneTimes, 1, 1);
      foreach (var timeZoneTime in timeZoneTimes)
         Console.WriteLine("{0}: {1:G}",
                           timeZoneTime.TimeZone == null ? "<null>" :
timeZoneTime.TimeZone.ToString(),
                           timeZoneTime.DateTime);
   }
}
// The example displays the following output:
//       (UTC-08:00) Pacific Time (US & Canada): 1/20/2014 12:11:00 PM
//       UTC: 1/20/2014 12:11:00 PM
//
//       (UTC-08:00) Pacific Time (US & Canada): 1/20/2014 12:11:00 PM
//       <null>: 1/1/0001 12:00:00 AM
```

Remarks

This method resets each element in an array to the element type's default value. It sets elements of reference types (including String elements) to `null`, and sets elements of value types to the default values shown in the following table.

Expand table

Type	Value
Boolean	`false`
All integral and floating point numeric types	0 (zero)
DateTime	DateTime.MinValue
Other value types	Default value of the type's fields

The range of cleared elements wrap from row to row in a multi-dimensional array.

This method only clears the values of the elements; it does not delete the elements themselves. An array has a fixed size; therefore, elements cannot be added or removed.

This method is an O(`n`) operation, where `n` is `length`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a shallow copy of the Array.

C#

```
public object Clone ();
```

Returns

Object

A shallow copy of the Array.

Implements

Clone()

Examples

The following code example clones a System.Globalization.CultureInfo array and demonstrates the behavior of a shallow copy.

C#

```
using System;
using System.Globalization;
public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new CultureInfo array.
      CultureInfo ci0 = new CultureInfo( "ar-SA", false );
      CultureInfo ci1 = new CultureInfo( "en-US", false );
      CultureInfo ci2 = new CultureInfo( "fr-FR", false );
      CultureInfo ci3 = new CultureInfo( "ja-JP", false );
      CultureInfo[] arrCI = new CultureInfo[] { ci0, ci1, ci2, ci3 };
```

```
      // Create a clone of the CultureInfo array.
      CultureInfo[] arrCIClone = (CultureInfo[]) arrCI.Clone();

      // Replace an element in the clone array.
      CultureInfo ci4 = new CultureInfo( "th-TH", false );
      arrCIClone[0] = ci4;

      // Display the contents of the original array.
      Console.WriteLine( "The original array contains the following values:"
);
      PrintIndexAndValues( arrCI );

      // Display the contents of the clone array.
      Console.WriteLine( "The clone array contains the following values:" );
      PrintIndexAndValues( arrCIClone );

      // Display the DateTimeFormatInfo.DateSeparator for the fourth element
in both arrays.
      Console.WriteLine( "Before changes to the clone:" );
      Console.WriteLine( "   Original: The DateTimeFormatInfo.DateSeparator
for {0} is {1}.", arrCI[3].Name, arrCI[3].DateTimeFormat.DateSeparator );
      Console.WriteLine( "      Clone: The DateTimeFormatInfo.DateSeparator
for {0} is {1}.", arrCIClone[3].Name,
arrCIClone[3].DateTimeFormat.DateSeparator );

      // Replace the DateTimeFormatInfo.DateSeparator for the fourth element
in the clone array.
      arrCIClone[3].DateTimeFormat.DateSeparator = "-";

      // Display the DateTimeFormatInfo.DateSeparator for the fourth element
in both arrays.
      Console.WriteLine( "After changes to the clone:" );
      Console.WriteLine( "   Original: The DateTimeFormatInfo.DateSeparator
for {0} is {1}.", arrCI[3].Name, arrCI[3].DateTimeFormat.DateSeparator );
      Console.WriteLine( "      Clone: The DateTimeFormatInfo.DateSeparator
for {0} is {1}.", arrCIClone[3].Name,
arrCIClone[3].DateTimeFormat.DateSeparator );
   }

   public static void PrintIndexAndValues( Array myArray )  {
      for ( int i = myArray.GetLowerBound(0); i <= myArray.GetUpperBound(0);
i++ )
         Console.WriteLine( "\t[{0}]:\t{1}", i, myArray.GetValue( i ) );
   }
}


/*
This code produces the following output.

The original array contains the following values:
        [0]:    ar-SA
        [1]:    en-US
        [2]:    fr-FR
```

```
        [3]:    ja-JP
The clone array contains the following values:
        [0]:    th-TH
        [1]:    en-US
        [2]:    fr-FR
        [3]:    ja-JP
Before changes to the clone:
   Original: The DateTimeFormatInfo.DateSeparator for ja-JP is /.
      Clone: The DateTimeFormatInfo.DateSeparator for ja-JP is /.
After changes to the clone:
   Original: The DateTimeFormatInfo.DateSeparator for ja-JP is -.
      Clone: The DateTimeFormatInfo.DateSeparator for ja-JP is -.

*/
```

Remarks

A shallow copy of an Array copies only the elements of the Array, whether they are reference types or value types, but it does not copy the objects that the references refer to. The references in the new Array point to the same objects that the references in the original Array point to.

In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

The clone is of the same Type as the original Array.

This method is an O(`n`) operation, where `n` is Length.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Copy
- ConstrainedCopy(Array, Int32, Array, Int32, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.ConstrainedCopy(Array, Int32, Array, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. Guarantees that all changes are undone if the copy does not succeed completely.

```
C#

public static void ConstrainedCopy (Array sourceArray, int sourceIndex,
Array destinationArray, int destinationIndex, int length);
```

Parameters

`sourceArray` Array

The Array that contains the data to copy.

`sourceIndex` Int32

A 32-bit integer that represents the index in the `sourceArray` at which copying begins.

`destinationArray` Array

The Array that receives the data.

`destinationIndex` Int32

A 32-bit integer that represents the index in the `destinationArray` at which storing begins.

`length` Int32

A 32-bit integer that represents the number of elements to copy.

Exceptions

ArgumentNullException

`sourceArray` is `null`.

-or-

`destinationArray` is `null`.

RankException

`sourceArray` and `destinationArray` have different ranks.

ArrayTypeMismatchException

The `sourceArray` type is neither the same as nor derived from the `destinationArray` type.

InvalidCastException

At least one element in `sourceArray` cannot be cast to the type of `destinationArray`.

ArgumentOutOfRangeException

`sourceIndex` is less than the lower bound of the first dimension of `sourceArray`.

-or-

`destinationIndex` is less than the lower bound of the first dimension of `destinationArray`.

-or-

`length` is less than zero.

ArgumentException

`length` is greater than the number of elements from `sourceIndex` to the end of `sourceArray`.

-or-

`length` is greater than the number of elements from `destinationIndex` to the end of `destinationArray`.

Remarks

The `sourceArray` and `destinationArray` parameters must have the same number of dimensions. The `sourceArray` type must be the same as or derived from the `destinationArray` type; otherwise, an ArrayTypeMismatchException is thrown. Unlike

Copy, ConstrainedCopy verifies the compatibility of the array types before performing any operation.

When copying between multidimensional arrays, the array behaves like a long one-dimensional array, where the rows (or columns) are conceptually laid end-to-end. For example, if an array has three rows (or columns) with four elements each, copying six elements from the beginning of the array would copy all four elements of the first row (or column) and the first two elements of the second row (or column). To start copying from the second element of the third row (or column), `sourceIndex` must be the upper bound of the first row (or column) plus the length of the second row (or column) plus two.

If `sourceArray` and `destinationArray` overlap, this method behaves as if the original values of `sourceArray` were preserved in a temporary location before `destinationArray` is overwritten.

[C++]

This method is equivalent to the standard C/C++ function `memmove`, not `memcpy`.

The arrays can be reference-type arrays or value-type arrays. If `sourceArray` and `destinationArray` are both reference-type arrays or are both arrays of type Object, a shallow copy is performed. A shallow copy of an Array is a new Array containing references to the same elements as the original Array. The elements themselves or anything referenced by the elements are not copied. In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

If this method throws an exception while copying, the `destinationArray` remains unchanged; therefore, ConstrainedCopy can be used within a constrained execution region (Cer).

This method is an O(`n`) operation, where `n` is `length`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Copy
- Clone()
- ReliabilityContractAttribute
- Cer

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.ConvertAll<TInput,TOutput> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts an array of one type to an array of another type.

C#

```
public static TOutput[] ConvertAll<TInput,TOutput> (TInput[] array,
Converter<TInput,TOutput> converter);
```

Type Parameters

TInput

The type of the elements of the source array.

TOutput

The type of the elements of the target array.

Parameters

array TInput[]

The one-dimensional, zero-based Array to convert to a target type.

converter Converter<TInput,TOutput>

A Converter<TInput,TOutput> that converts each element from one type to another type.

Returns

TOutput[]

An array of the target type containing the converted elements from the source array.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`converter` is `null`.

Examples

The following code example defines a method named `PointFToPoint` that converts a PointF structure to a Point structure. The example then creates an array of PointF structures, creates a `Converter<PointF, Point>` delegate (`Converter(Of PointF, Point)` in Visual Basic) to represent the `PointFToPoint` method, and passes the delegate to the ConvertAll method. The ConvertAll method passes each element of the input list to the `PointFToPoint` method and puts the converted elements into a new list of Point structures. In the F# example, the `pointFToPoint` function is implicitly casted to the `Converter<PointF, Point>` delegate. Both lists are displayed.

C#

```
using System;
using System.Drawing;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        // Create an array of PointF objects.
        PointF[] apf = {
            new PointF(27.8F, 32.62F),
            new PointF(99.3F, 147.273F),
            new PointF(7.5F, 1412.2F) };

        // Display each element in the PointF array.
        Console.WriteLine();
        foreach( PointF p in apf )
            Console.WriteLine(p);

        // Convert each PointF element to a Point object.
        Point[] ap = Array.ConvertAll(apf,
            new Converter<PointF, Point>(PointFToPoint));

        // Display each element in the Point array.
        Console.WriteLine();
        foreach( Point p in ap )
```

```
        {
            Console.WriteLine(p);
        }
    }

    public static Point PointFToPoint(PointF pf)
    {
        return new Point(((int) pf.X), ((int) pf.Y));
    }
}

/* This code example produces the following output:

{X=27.8, Y=32.62}
{X=99.3, Y=147.273}
{X=7.5, Y=1412.2}

{X=27,Y=32}
{X=99,Y=147}
{X=7,Y=1412}
 */
```

Remarks

The Converter<TInput,TOutput> is a delegate to a method that converts an object to the target type. The elements of `array` are individually passed to the Converter<TInput,TOutput>, and the converted elements are saved in the new array.

The source `array` remains unchanged.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In F#, the Array.map function is generally used instead.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Converter<TInput,TOutput>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Copy Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies a range of elements in one Array to another Array and performs type casting and boxing as required.

Overloads

Expand table

Copy(Array, Int64, Array, Int64, Int64)	Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 64-bit integers.
Copy(Array, Int32, Array, Int32, Int32)	Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 32-bit integers.
Copy(Array, Array, Int32)	Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
Copy(Array, Array, Int64)	Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 64-bit integer.

Examples

The following code example shows how to copy from one Array of type Object to another Array of type integer.

C#

```
using System;
public class SamplesArray  {

   public static void Main()  {
```

```
      // Creates and initializes a new Array of type int.
      Array myIntArray=Array.CreateInstance( typeof(System.Int32), 5 );
      for ( int i = myIntArray.GetLowerBound(0); i <=
myIntArray.GetUpperBound(0); i++ )
         myIntArray.SetValue( i+1, i );

      // Creates and initializes a new Array of type Object.
      Array myObjArray = Array.CreateInstance( typeof(System.Object), 5 );
      for ( int i = myObjArray.GetLowerBound(0); i <=
myObjArray.GetUpperBound(0); i++ )
         myObjArray.SetValue( i+26, i );

      // Displays the initial values of both arrays.
      Console.WriteLine( "int array:" );
      PrintValues( myIntArray );
      Console.WriteLine( "Object array:" );
      PrintValues( myObjArray );

      // Copies the first element from the int array to the Object array.
      Array.Copy( myIntArray, myIntArray.GetLowerBound(0), myObjArray,
myObjArray.GetLowerBound(0), 1 );

      // Copies the last two elements from the Object array to the int
array.
      Array.Copy( myObjArray, myObjArray.GetUpperBound(0) - 1, myIntArray,
myIntArray.GetUpperBound(0) - 1, 2 );

      // Displays the values of the modified arrays.
      Console.WriteLine( "int array - Last two elements should now be the
same as Object array:" );
      PrintValues( myIntArray );
      Console.WriteLine( "Object array - First element should now be the
same as int array:" );
      PrintValues( myObjArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator = myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
         } else  {
            Console.WriteLine();
            i = 1;
         }
         Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
   }
}
/*
This code produces the following output.
```

```
int array:
    1    2    3    4    5
Object array:
    26    27    28    29    30
int array - Last two elements should now be the same as Object array:
    1    2    3    29    30
Object array - First element should now be the same as int array:
    1    27    28    29    30
*/
```

Copy(Array, Int64, Array, Int64, Int64)

Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 64-bit integers.

C#

```
public static void Copy (Array sourceArray, long sourceIndex, Array
destinationArray, long destinationIndex, long length);
```

Parameters

`sourceArray` Array

The Array that contains the data to copy.

`sourceIndex` Int64

A 64-bit integer that represents the index in the `sourceArray` at which copying begins.

`destinationArray` Array

The Array that receives the data.

`destinationIndex` Int64

A 64-bit integer that represents the index in the `destinationArray` at which storing begins.

`length` Int64

A 64-bit integer that represents the number of elements to copy. The integer must be between zero and Int32.MaxValue, inclusive.

Exceptions

ArgumentNullException

`sourceArray` is `null`.

-or-

`destinationArray` is `null`.

RankException

`sourceArray` and `destinationArray` have different ranks.

ArrayTypeMismatchException

`sourceArray` and `destinationArray` are of incompatible types.

InvalidCastException

At least one element in `sourceArray` cannot be cast to the type of `destinationArray`.

ArgumentOutOfRangeException

`sourceIndex` is outside the range of valid indexes for the `sourceArray`.

-or-

`destinationIndex` is outside the range of valid indexes for the `destinationArray`.

-or-

`length` is less than 0 or greater than Int32.MaxValue.

ArgumentException

`length` is greater than the number of elements from `sourceIndex` to the end of `sourceArray`.

-or-

`length` is greater than the number of elements from `destinationIndex` to the end of `destinationArray`.

Remarks

The `sourceArray` and `destinationArray` parameters must have the same number of dimensions. In addition, `destinationArray` must already have been dimensioned and

must have a sufficient number of elements starting from the `destinationIndex` position to accommodate the copied data.

When copying between multidimensional arrays, the array behaves like a long one-dimensional array, where the rows (or columns) are conceptually laid end-to-end. For example, if an array has three rows (or columns) with four elements each, copying six elements from the beginning of the array would copy all four elements of the first row (or column) and the first two elements of the second row (or column). To start copying from the second element of the third row (or column), `sourceIndex` must be the upper bound of the first row (or column) plus the length of the second row (or column) plus two.

If `sourceArray` and `destinationArray` overlap, this method behaves as if the original values of `sourceArray` were preserved in a temporary location before `destinationArray` is overwritten.

[C++]

This method is equivalent to the standard C/C++ function `memmove`, not `memcpy`.

The arrays can be reference-type arrays or value-type arrays. Type downcasting is performed, as required.

- When copying from a reference-type array to a value-type array, each element is unboxed and then copied. When copying from a value-type array to a reference-type array, each element is boxed and then copied.
- When copying from a reference-type or value-type array to an Object array, an Object is created to hold each value or reference and then copied. When copying from an Object array to a reference-type or value-type array and the assignment is not possible, an InvalidCastException is thrown.
- If `sourceArray` and `destinationArray` are both reference-type arrays or are both arrays of type Object, a shallow copy is performed. A shallow copy of an Array is a new Array containing references to the same elements as the original Array. The elements themselves or anything referenced by the elements are not copied. In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

An ArrayTypeMismatchException is thrown if the arrays are of incompatible types. Type compatibility is defined as follows:

- A type is compatible with itself.

- A value type is compatible with Object and with an interface type implemented by that value type. A value type is considered connected to an interface only if it implements that interface directly. Disconnected types are not compatible.
- Two intrinsic (predefined) value types are compatible if copying from the source type to the destination type is a widening conversion. A widening conversion never loses information, whereas a narrowing conversion can lose information. For example, converting a 32-bit signed integer to a 64-bit signed integer is a widening conversion, and converting a 64-bit signed integer to a 32-bit signed integer is a narrowing conversion. For more information about conversions, see Convert.
- A nonintrinsic (user-defined) value type is compatible only with itself.
- Enumerations have an implicit conversion to Enum and to their underlying type.

If every element in `sourceArray` requires a downcast (for example, from a base class to a derived class or from an interface to an object) and one or more elements cannot be cast to the corresponding type in `destinationArray`, an InvalidCastException is thrown.

If this method throws an exception while copying, the state of `destinationArray` is undefined.

This method is an O(`n`) operation, where `n` is `length`.

See also

- ConstrainedCopy(Array, Int32, Array, Int32, Int32)
- Clone()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Copy(Array, Int32, Array, Int32, Int32)

Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. The length and the indexes are specified as 32-bit integers.

```
C#

public static void Copy (Array sourceArray, int sourceIndex, Array
destinationArray, int destinationIndex, int length);
```

Parameters

`sourceArray` Array

The Array that contains the data to copy.

`sourceIndex` Int32

A 32-bit integer that represents the index in the `sourceArray` at which copying begins.

`destinationArray` Array

The Array that receives the data.

`destinationIndex` Int32

A 32-bit integer that represents the index in the `destinationArray` at which storing begins.

`length` Int32

A 32-bit integer that represents the number of elements to copy.

Exceptions

ArgumentNullException

`sourceArray` is `null`.

-or-

`destinationArray` is `null`.

RankException

`sourceArray` and `destinationArray` have different ranks.

ArrayTypeMismatchException

`sourceArray` and `destinationArray` are of incompatible types.

InvalidCastException

At least one element in `sourceArray` cannot be cast to the type of `destinationArray`.

ArgumentOutOfRangeException

`sourceIndex` is less than the lower bound of the first dimension of `sourceArray`.

-or-

`destinationIndex` is less than the lower bound of the first dimension of `destinationArray`.

-or-

`length` is less than zero.

ArgumentException

`length` is greater than the number of elements from `sourceIndex` to the end of `sourceArray`.

-or-

`length` is greater than the number of elements from `destinationIndex` to the end of `destinationArray`.

Remarks

The `sourceArray` and `destinationArray` parameters must have the same number of dimensions. In addition, `destinationArray` must already have been dimensioned and must have a sufficient number of elements starting from the `destinationIndex` position to accommodate the copied data.

When copying between multidimensional arrays, the array behaves like a long one-dimensional array, where the rows (or columns) are conceptually laid end-to-end. For example, if an array has three rows (or columns) with four elements each, copying six elements from the beginning of the array would copy all four elements of the first row (or column) and the first two elements of the second row (or column). To start copying from the second element of the third row (or column), `sourceIndex` must be the upper bound of the first row (or column) plus the length of the second row (or column) plus two.

If `sourceArray` and `destinationArray` overlap, this method behaves as if the original values of `sourceArray` were preserved in a temporary location before `destinationArray` is overwritten.

[C++]

This method is equivalent to the standard C/C++ function `memmove`, not `memcpy`.

The arrays can be reference-type arrays or value-type arrays. Type downcasting is performed, as required.

- When copying from a reference-type array to a value-type array, each element is unboxed and then copied. When copying from a value-type array to a reference-type array, each element is boxed and then copied.
- When copying from a reference-type or value-type array to an Object array, an Object is created to hold each value or reference and then copied. When copying from an Object array to a reference-type or value-type array and the assignment is not possible, an InvalidCastException is thrown.
- If `sourceArray` and `destinationArray` are both reference-type arrays or are both arrays of type Object, a shallow copy is performed. A shallow copy of an Array is a new Array containing references to the same elements as the original Array. The elements themselves or anything referenced by the elements are not copied. In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

An ArrayTypeMismatchException is thrown if the arrays are of incompatible types. Type compatibility is defined as follows:

- A type is compatible with itself.
- A value type is compatible with Object and with an interface type implemented by that value type. A value type is considered connected to an interface only if it implements that interface directly. Disconnected types are not compatible.
- Two intrinsic (predefined) value types are compatible if copying from the source type to the destination type is a widening conversion. A widening conversion never loses information, whereas a narrowing conversion can lose information. For example, converting a 32-bit signed integer to a 64-bit signed integer is a widening conversion, and converting a 64-bit signed integer to a 32-bit signed integer is a narrowing conversion. For more information about conversions, see Convert.

- A nonintrinsic (user-defined) value type is compatible only with itself.
- Enumerations have an implicit conversion to Enum and to their underlying type.

If every element in `sourceArray` requires a downcast (for example, from a base class to a derived class or from an interface to an object) and one or more elements cannot be cast to the corresponding type in `destinationArray`, an InvalidCastException is thrown.

If this method throws an exception while copying, the state of `destinationArray` is undefined.

This method is an O(`n`) operation, where `n` is `length`.

See also

- ConstrainedCopy(Array, Int32, Array, Int32, Int32)
- Clone()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Copy(Array, Array, Int32)

Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.

```
C#
```

```
public static void Copy (Array sourceArray, Array destinationArray, int
length);
```

Parameters

`sourceArray` Array

The Array that contains the data to copy.

`destinationArray` Array

The Array that receives the data.

`length` Int32

A 32-bit integer that represents the number of elements to copy.

Exceptions

ArgumentNullException

`sourceArray` is `null`.

-or-

`destinationArray` is `null`.

RankException

`sourceArray` and `destinationArray` have different ranks.

ArrayTypeMismatchException

`sourceArray` and `destinationArray` are of incompatible types.

InvalidCastException

At least one element in `sourceArray` cannot be cast to the type of `destinationArray`.

ArgumentOutOfRangeException

`length` is less than zero.

ArgumentException

`length` is greater than the number of elements in `sourceArray`.

-or-

`length` is greater than the number of elements in `destinationArray`.

Remarks

The `sourceArray` and `destinationArray` parameters must have the same number of dimensions. In addition, `destinationArray` must already have been dimensioned and must have a sufficient number of elements to accommodate the copied data.

When copying between multidimensional arrays, the array behaves like a long one-dimensional array, where the rows (or columns) are conceptually laid end to end. For example, if an array has three rows (or columns) with four elements each, copying six elements from the beginning of the array would copy all four elements of the first row (or column) and the first two elements of the second row (or column).

If `sourceArray` and `destinationArray` overlap, this method behaves as if the original values of `sourceArray` were preserved in a temporary location before `destinationArray` is overwritten.

[C++]

This method is equivalent to the standard C/C++ function `memmove`, not `memcpy`.

The arrays can be reference-type arrays or value-type arrays. Type downcasting is performed, as required.

- When copying from a reference-type array to a value-type array, each element is unboxed and then copied. When copying from a value-type array to a reference-type array, each element is boxed and then copied.

- When copying from a reference-type or value-type array to an Object array, an Object is created to hold each value or reference and then copied. When copying from an Object array to a reference-type or value-type array and the assignment is not possible, an InvalidCastException is thrown.

- If `sourceArray` and `destinationArray` are both reference-type arrays or are both arrays of type Object, a shallow copy is performed. A shallow copy of an Array is a new Array containing references to the same elements as the original Array. The elements themselves or anything referenced by the elements are not copied. In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

An ArrayTypeMismatchException is thrown if the arrays are of incompatible types. Type compatibility is defined as follows:

- A type is compatible with itself.

- A value type is compatible with Object and with an interface type implemented by that value type. A value type is considered connected to an interface only if it implements that interface directly. Disconnected types are not compatible.
- Two intrinsic (predefined) value types are compatible if copying from the source type to the destination type is a widening conversion. A widening conversion never loses information, whereas a narrowing conversion can lose information. For example, converting a 32-bit signed integer to a 64-bit signed integer is a widening conversion, and converting a 64-bit signed integer to a 32-bit signed integer is a narrowing conversion. For more information about conversions, see Convert.
- A nonintrinsic (user-defined) value type is compatible only with itself.
- Enumerations have an implicit conversion to Enum and to their underlying type.

If every element in `sourceArray` requires a downcast (for example, from a base class to a derived class or from an interface to an object) and one or more elements cannot be cast to the corresponding type in `destinationArray`, an InvalidCastException is thrown.

If this method throws an exception while copying, the state of `destinationArray` is undefined.

This method is an O(`n`) operation, where `n` is `length`.

See also

- ConstrainedCopy(Array, Int32, Array, Int32, Int32)
- Clone()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Copy(Array, Array, Int64)

Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 64-bit integer.

```
C#

public static void Copy (Array sourceArray, Array destinationArray, long
length);
```

Parameters

`sourceArray` Array

The Array that contains the data to copy.

`destinationArray` Array

The Array that receives the data.

`length` Int64

A 64-bit integer that represents the number of elements to copy. The integer must be between zero and Int32.MaxValue, inclusive.

Exceptions

ArgumentNullException

`sourceArray` is `null`.

-or-

`destinationArray` is `null`.

RankException

`sourceArray` and `destinationArray` have different ranks.

ArrayTypeMismatchException

`sourceArray` and `destinationArray` are of incompatible types.

InvalidCastException

At least one element in `sourceArray` cannot be cast to the type of `destinationArray`.

ArgumentOutOfRangeException

`length` is less than 0 or greater than Int32.MaxValue.

ArgumentException

`length` is greater than the number of elements in `sourceArray`.

-or-

`length` is greater than the number of elements in `destinationArray`.

Remarks

The `sourceArray` and `destinationArray` parameters must have the same number of dimensions. In addition, `destinationArray` must already have been dimensioned and must have a sufficient number of elements to accommodate the copied data.

When copying between multidimensional arrays, the array behaves like a long one-dimensional array, where the rows (or columns) are conceptually laid end to end. For example, if an array has three rows (or columns) with four elements each, copying six elements from the beginning of the array would copy all four elements of the first row (or column) and the first two elements of the second row (or column).

If `sourceArray` and `destinationArray` overlap, this method behaves as if the original values of `sourceArray` were preserved in a temporary location before `destinationArray` is overwritten.

[C++]

This method is equivalent to the standard C/C++ function `memmove`, not `memcpy`.

The arrays can be reference-type arrays or value-type arrays. Type downcasting is performed, as required.

- When copying from a reference-type array to a value-type array, each element is unboxed and then copied. When copying from a value-type array to a reference-type array, each element is boxed and then copied.
- When copying from a reference-type or value-type array to an Object array, an Object is created to hold each value or reference and then copied. When

copying from an Object array to a reference-type or value-type array and the assignment is not possible, an InvalidCastException is thrown.

- If `sourceArray` and `destinationArray` are both reference-type arrays or are both arrays of type Object, a shallow copy is performed. A shallow copy of an Array is a new Array containing references to the same elements as the original Array. The elements themselves or anything referenced by the elements are not copied. In contrast, a deep copy of an Array copies the elements and everything directly or indirectly referenced by the elements.

An ArrayTypeMismatchException is thrown if the arrays are of incompatible types. Type compatibility is defined as follows:

- A type is compatible with itself.
- A value type is compatible with Object and with an interface type implemented by that value type. A value type is considered connected to an interface only if it implements that interface directly. Disconnected types are not compatible.
- Two intrinsic (predefined) value types are compatible if copying from the source type to the destination type is a widening conversion. A widening conversion never loses information, whereas a narrowing conversion can lose information. For example, converting a 32-bit signed integer to a 64-bit signed integer is a widening conversion, and converting a 64-bit signed integer to a 32-bit signed integer is a narrowing conversion. For more information about conversions, see Convert.
- A nonintrinsic (user-defined) value type is compatible only with itself.
- Enumerations have an implicit conversion to Enum and to their underlying type.

If every element in `sourceArray` requires a downcast (for example, from a base class to a derived class or from an interface to an object) and one or more elements cannot be cast to the corresponding type in `destinationArray`, an InvalidCastException is thrown.

If this method throws an exception while copying, the state of `destinationArray` is undefined.

This method is an O(`n`) operation, where `n` is `length`.

See also

- ConstrainedCopy(Array, Int32, Array, Int32, Int32)

- Clone()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.CopyTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies all the elements of the current one-dimensional array to the specified one-dimensional array.

Overloads

Expand table

CopyTo(Array, Int32)	Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 32-bit integer.
CopyTo(Array, Int64)	Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 64-bit integer.

Examples

The following code example shows how to copy an Array to another Array.

C#

```
using System;

public class SamplesArray
{

   public static void Main()
   {

      // Creates and initializes two new Arrays.
      Array mySourceArray=Array.CreateInstance(typeof(string), 6);
      mySourceArray.SetValue("three", 0);
      mySourceArray.SetValue("napping", 1);
      mySourceArray.SetValue("cats", 2);
      mySourceArray.SetValue("in", 3);
      mySourceArray.SetValue("the", 4);
```

```
      mySourceArray.SetValue("barn", 5);
      Array myTargetArray=Array.CreateInstance(typeof(string), 15);
      myTargetArray.SetValue("The", 0);
      myTargetArray.SetValue("quick", 1);
      myTargetArray.SetValue("brown", 2);
      myTargetArray.SetValue("fox", 3);
      myTargetArray.SetValue("jumps", 4);
      myTargetArray.SetValue("over", 5);
      myTargetArray.SetValue("the", 6);
      myTargetArray.SetValue("lazy", 7);
      myTargetArray.SetValue("dog", 8);

      // Displays the values of the Array.
      Console.WriteLine("The target Array contains the following (before and
after copying):");
      PrintValues(myTargetArray, ' ');

      // Copies the source Array to the target Array, starting at index 6.
      mySourceArray.CopyTo(myTargetArray, 6);

      // Displays the values of the Array.
      PrintValues(myTargetArray, ' ');
   }

   public static void PrintValues(Array myArr, char mySeparator)
   {

      System.Collections.IEnumerator myEnumerator = myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength(myArr.Rank - 1);
      while (myEnumerator.MoveNext())
      {
         if (i < cols)
         {
            i++;
         }
         else
         {
             Console.WriteLine();
             i = 1;
         }
         Console.Write("{0}{1}", mySeparator, myEnumerator.Current);
      }
      Console.WriteLine();
   }
}
/*
This code produces the following output.

The target Array contains the following (before and after copying):
 The quick brown fox jumps over the lazy dog
 The quick brown fox jumps over three napping cats in the barn
*/
```

The following code example shows how to copy an Array to another Array with a nonzero lower bound. Note that the entire source Array is copied, including empty elements that overwrite existing elements in the target Array.

C#

```
using System;

public class SamplesArray2
{

   public static void Main()
   {
      // Creates and initializes the source Array.
      Array myArrayZero=Array.CreateInstance(typeof(string), 3);
      myArrayZero.SetValue("zero", 0);
      myArrayZero.SetValue("one", 1);

      // Displays the source Array.
      Console.WriteLine("The array with lower bound=0 contains:");
      PrintIndexAndValues(myArrayZero);

      // Creates and initializes the target Array.
      int[] myArrLen = { 4 };
      int[] myArrLow = { 2 };
      Array myArrayTwo=Array.CreateInstance(typeof(string), myArrLen,
myArrLow);
      myArrayTwo.SetValue("two", 2);
      myArrayTwo.SetValue("three", 3);
      myArrayTwo.SetValue("four", 4);
      myArrayTwo.SetValue("five", 5);

      // Displays the target Array.
      Console.WriteLine("The array with lower bound=2 contains:");
      PrintIndexAndValues(myArrayTwo);

      // Copies from the array with lower bound=0 to the array with lower
bound=2.
      myArrayZero.CopyTo(myArrayTwo, 3);

      // Displays the modified target Array.
      Console.WriteLine("\nAfter copying to the target array from index
3:");
      PrintIndexAndValues(myArrayTwo);
   }

   public static void PrintIndexAndValues(Array myArray)
   {
      for (int i = myArray.GetLowerBound(0); i <= myArray.GetUpperBound(0);
i++)
         Console.WriteLine("\t[{0}]:\t{1}", i, myArray.GetValue(i));
   }
}
```

```
/*
This code produces the following output.

The array with lower bound=0 contains:
    [0]:    zero
    [1]:    one
    [2]:
The array with lower bound=2 contains:
    [2]:    two
    [3]:    three
    [4]:    four
    [5]:    five

After copying to the target array from index 3:
    [2]:    two
    [3]:    zero
    [4]:    one
    [5]:
*/
```

CopyTo(Array, Int32)

Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 32-bit integer.

```
C#

public void CopyTo (Array array, int index);
```

Parameters

`array` Array

The one-dimensional array that is the destination of the elements copied from the current array.

`index` Int32

A 32-bit integer that represents the index in `array` at which copying begins.

Implements

CopyTo(Array, Int32)

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

ArgumentException

`array` is multidimensional.

-or-

The number of elements in the source array is greater than the available number of elements from `index` to the end of the destination `array`.

ArrayTypeMismatchException

The type of the source Array cannot be cast automatically to the type of the destination `array`.

RankException

The source array is multidimensional.

InvalidCastException

At least one element in the source Array cannot be cast to the type of destination `array`.

Remarks

This method copies all the elements of the current array instance to the `array` destination array, starting at index `index`. The `array` destination array must already have been dimensioned and must have a sufficient number of elements to accommodate the copied elements. Otherwise, the method throws an exception.

This method supports the System.Collections.ICollection interface. If implementing System.Collections.ICollection is not explicitly required, use Copy to avoid an extra indirection.

If this method throws an exception while copying, the state of `array` is undefined.

This method is an O(`n`) operation, where `n` is Length. It performs a shallow copy only.

See also

- ICollection
- Copy(Array, Array, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CopyTo(Array, Int64)

Copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index. The index is specified as a 64-bit integer.

C#

```
public void CopyTo (Array array, long index);
```

Parameters

`array` Array

The one-dimensional array that is the destination of the elements copied from the current array.

`index` Int64

A 64-bit integer that represents the index in `array` at which copying begins.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is outside the range of valid indexes for `array`.

ArgumentException

`array` is multidimensional.

-or-

The number of elements in the source array is greater than the available number of elements from `index` to the end of the destination `array`.

ArrayTypeMismatchException

The type of the source Array cannot be cast automatically to the type of the destination `array`.

RankException

The source Array is multidimensional.

InvalidCastException

At least one element in the source Array cannot be cast to the type of destination `array`.

Remarks

This method copies all the elements of the current array instance to the `array` destination array, starting at index `index`. The `array` destination array must already have been dimensioned and must have a sufficient number of elements to accommodate the copied elements. Otherwise, the method throws an exception.

This method supports the System.Collections.ICollection interface. If implementing System.Collections.ICollection is not explicitly required, use Copy to avoid an extra indirection.

If this method throws an exception while copying, the state of `array` is undefined.

This method is an O(`n`) operation, where `n` is Length. It performs a shallow copy only.

See also

- ICollection
- Copy(Array, Array, Int64)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.CreateInstance Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Array class.

Overloads

Expand table

CreateInstance(Type, Int32)	Creates a one-dimensional Array of the specified Type and length, with zero-based indexing.
CreateInstance(Type, Int32[])	Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 32-bit integers.
CreateInstance(Type, Int64[])	Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 64-bit integers.
CreateInstance(Type, Int32, Int32)	Creates a two-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.
CreateInstance(Type, Int32[], Int32[])	Creates a multidimensional Array of the specified Type and dimension lengths, with the specified lower bounds.
CreateInstance(Type, Int32, Int32, Int32)	Creates a three-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.

CreateInstance(Type, Int32)

Creates a one-dimensional Array of the specified Type and length, with zero-based indexing.

C#

```
public static Array CreateInstance (Type elementType, int length);
```

Parameters

`elementType` Type

The Type of the Array to create.

`length` Int32

The size of the Array to create.

Returns

Array

A new one-dimensional Array of the specified Type with the specified length, using zero-based indexing.

Exceptions

ArgumentNullException

`elementType` is `null`.

ArgumentException

`elementType` is not a valid Type.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

`length` is less than zero.

Examples

The following code example shows how to create and initialize a one-dimensional Array.

C#

```
using System;
public class SamplesArray  {

   public static void Main()  {
```

```
      // Creates and initializes a one-dimensional Array of type int.
      Array my1DArray=Array.CreateInstance( typeof(int), 5 );
      for ( int i = my1DArray.GetLowerBound(0); i <=
my1DArray.GetUpperBound(0); i++ )
         my1DArray.SetValue( i+1, i );

      // Displays the values of the Array.
      Console.WriteLine( "The one-dimensional Array contains the
following values:" );
      PrintValues( my1DArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
         } else  {
            Console.WriteLine();
            i = 1;
         }
         Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
   }
}
/*
This code produces the following output.

The one-dimensional Array contains the following values:
    1    2    3    4    5
*/
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is `length`.

In F#, the Array.zeroCreate function is usually used instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CreateInstance(Type, Int32[])

Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 32-bit integers.

C#

```
public static Array CreateInstance (Type elementType, params int[]
lengths);
```

Parameters

`elementType` Type

The Type of the Array to create.

`lengths` Int32[]

An array of 32-bit integers that represent the size of each dimension of the Array to create.

Returns

Array

A new multidimensional Array of the specified Type with the specified length for each dimension, using zero-based indexing.

Exceptions

ArgumentNullException

`elementType` is `null`.

-or-

`lengths` is `null`.

ArgumentException

`elementType` is not a valid Type.

-or-

The `lengths` array contains less than one element.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

Any value in `lengths` is less than zero.

Examples

The following code example shows how to create and initialize a multidimensional Array.

C#

```
using System;
public class SamplesArray3  {

   public static void Main()  {

      // Creates and initializes a multidimensional Array of type string.
      int[] myLengthsArray = new int[4] { 2, 3, 4, 5 };
      Array my4DArray=Array.CreateInstance( typeof(string),
myLengthsArray );
      for ( int i = my4DArray.GetLowerBound(0); i <=
my4DArray.GetUpperBound(0); i++ )
         for ( int j = my4DArray.GetLowerBound(1); j <=
my4DArray.GetUpperBound(1); j++ )
            for ( int k = my4DArray.GetLowerBound(2); k <=
my4DArray.GetUpperBound(2); k++ )
               for ( int l = my4DArray.GetLowerBound(3); l <=
```

```
my4DArray.GetUpperBound(3); l++ )  {
                  int[] myIndicesArray = new int[4] { i, j, k, l };
                  my4DArray.SetValue( Convert.ToString(i) + j + k + l,
myIndicesArray );
               }

      // Displays the values of the Array.
      Console.WriteLine( "The four-dimensional Array contains the
following values:" );
      PrintValues( my4DArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
         } else  {
            Console.WriteLine();
            i = 1;
         }
         Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
   }
}
/*
This code produces the following output.

The four-dimensional Array contains the following values:
    0000    0001    0002    0003    0004
    0010    0011    0012    0013    0014
    0020    0021    0022    0023    0024
    0030    0031    0032    0033    0034
    0100    0101    0102    0103    0104
    0110    0111    0112    0113    0114
    0120    0121    0122    0123    0124
    0130    0131    0132    0133    0134
    0200    0201    0202    0203    0204
    0210    0211    0212    0213    0214
    0220    0221    0222    0223    0224
    0230    0231    0232    0233    0234
    1000    1001    1002    1003    1004
    1010    1011    1012    1013    1014
    1020    1021    1022    1023    1024
    1030    1031    1032    1033    1034
    1100    1101    1102    1103    1104
    1110    1111    1112    1113    1114
    1120    1121    1122    1123    1124
    1130    1131    1132    1133    1134
    1200    1201    1202    1203    1204
    1210    1211    1212    1213    1214
```

```
    1220    1221    1222    1223    1224
    1230    1231    1232    1233    1234
*/
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

The number of elements in the `lengths` array must equal the number of dimensions in the new Array. Each element of the `lengths` array must specify the length of the corresponding dimension in the new Array.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is the product of all values in `lengths`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CreateInstance(Type, Int64[])

Creates a multidimensional Array of the specified Type and dimension lengths, with zero-based indexing. The dimension lengths are specified in an array of 64-bit integers.

C#

```
public static Array CreateInstance (Type elementType, params long[]
```

```
lengths);
```

Parameters

`elementType` Type

The Type of the Array to create.

`lengths` Int64[]

An array of 64-bit integers that represent the size of each dimension of the Array to create. Each integer in the array must be between zero and Int32.MaxValue, inclusive.

Returns

Array

A new multidimensional Array of the specified Type with the specified length for each dimension, using zero-based indexing.

Exceptions

ArgumentNullException

`elementType` is `null`.

-or-

`lengths` is `null`.

ArgumentException

`elementType` is not a valid Type.

-or-

The `lengths` array contains less than one element.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

Any value in `lengths` is less than zero or greater than Int32.MaxValue.

Examples

The following code example shows how to create and initialize a multidimensional Array.

C#

```
using System;
public class SamplesArray3  {

   public static void Main()  {

      // Creates and initializes a multidimensional Array of type string.
      int[] myLengthsArray = new int[4] { 2, 3, 4, 5 };
      Array my4DArray=Array.CreateInstance( typeof(string),
myLengthsArray );
      for ( int i = my4DArray.GetLowerBound(0); i <=
my4DArray.GetUpperBound(0); i++ )
         for ( int j = my4DArray.GetLowerBound(1); j <=
my4DArray.GetUpperBound(1); j++ )
            for ( int k = my4DArray.GetLowerBound(2); k <=
my4DArray.GetUpperBound(2); k++ )
               for ( int l = my4DArray.GetLowerBound(3); l <=
my4DArray.GetUpperBound(3); l++ )  {
                  int[] myIndicesArray = new int[4] { i, j, k, l };
                  my4DArray.SetValue( Convert.ToString(i) + j + k + l,
myIndicesArray );
               }

      // Displays the values of the Array.
      Console.WriteLine( "The four-dimensional Array contains the
following values:" );
      PrintValues( my4DArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
         } else  {
            Console.WriteLine();
            i = 1;
         }
         Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
   }
}
/*
```

```
This code produces the following output.

The four-dimensional Array contains the following values:
    0000    0001    0002    0003    0004
    0010    0011    0012    0013    0014
    0020    0021    0022    0023    0024
    0030    0031    0032    0033    0034
    0100    0101    0102    0103    0104
    0110    0111    0112    0113    0114
    0120    0121    0122    0123    0124
    0130    0131    0132    0133    0134
    0200    0201    0202    0203    0204
    0210    0211    0212    0213    0214
    0220    0221    0222    0223    0224
    0230    0231    0232    0233    0234
    1000    1001    1002    1003    1004
    1010    1011    1012    1013    1014
    1020    1021    1022    1023    1024
    1030    1031    1032    1033    1034
    1100    1101    1102    1103    1104
    1110    1111    1112    1113    1114
    1120    1121    1122    1123    1124
    1130    1131    1132    1133    1134
    1200    1201    1202    1203    1204
    1210    1211    1212    1213    1214
    1220    1221    1222    1223    1224
    1230    1231    1232    1233    1234
*/
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

The number of elements in the `lengths` array must equal the number of dimensions in the new Array. Each element of the `lengths` array must specify the length of the corresponding dimension in the new Array.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is the product of all values in `lengths`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateInstance(Type, Int32, Int32)

Creates a two-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.

C#

```
public static Array CreateInstance (Type elementType, int length1, int
length2);
```

Parameters

`elementType` Type

The Type of the Array to create.

`length1` Int32

The size of the first dimension of the Array to create.

`length2` Int32

The size of the second dimension of the Array to create.

Returns

Array

A new two-dimensional Array of the specified Type with the specified length for each dimension, using zero-based indexing.

Exceptions

ArgumentNullException

`elementType` is `null`.

ArgumentException

`elementType` is not a valid Type.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

`length1` is less than zero.

-or-

`length2` is less than zero.

Examples

The following code example shows how to create and initialize a two-dimensional Array.

C#

```
using System;
public class SamplesArray1  {

   public static void Main()  {

      // Creates and initializes a two-dimensional Array of type string.
      Array my2DArray=Array.CreateInstance( typeof(string), 2, 3 );
      for ( int i = my2DArray.GetLowerBound(0); i <=
my2DArray.GetUpperBound(0); i++ )
         for ( int j = my2DArray.GetLowerBound(1); j <=
my2DArray.GetUpperBound(1); j++ )
            my2DArray.SetValue( "abc" + i + j, i, j );

      // Displays the values of the Array.
      Console.WriteLine( "The two-dimensional Array contains the
following values:" );
      PrintValues( my2DArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
```

```
        while ( myEnumerator.MoveNext() )  {
           if ( i < cols )  {
              i++;
           } else  {
              Console.WriteLine();
              i = 1;
           }
           Console.Write( "\t{0}", myEnumerator.Current );
        }
        Console.WriteLine();
    }
}
/*
This code produces the following output.

The two-dimensional Array contains the following values:
    abc00    abc01    abc02
    abc10    abc11    abc12
*/
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is the product of `length1` and `length2`.

In F#, the Array2D.zeroCreate function can be used instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateInstance(Type, Int32[], Int32[])

Creates a multidimensional Array of the specified Type and dimension lengths, with the specified lower bounds.

```
C#

public static Array CreateInstance (Type elementType, int[] lengths,
int[] lowerBounds);
```

Parameters

`elementType` Type

The Type of the Array to create.

`lengths` Int32[]

A one-dimensional array that contains the size of each dimension of the Array to create.

`lowerBounds` Int32[]

A one-dimensional array that contains the lower bound (starting index) of each dimension of the Array to create.

Returns

Array

A new multidimensional Array of the specified Type with the specified length and lower bound for each dimension.

Exceptions

ArgumentNullException

`elementType` is `null`.

-or-

`lengths` is `null`.

-or-

`lowerBounds` is `null`.

ArgumentException

`elementType` is not a valid Type.

-or-

The `lengths` array contains less than one element.

-or-

The `lengths` and `lowerBounds` arrays do not contain the same number of elements.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

Any value in `lengths` is less than zero.

-or-

Any value in `lowerBounds` is very large, such that the sum of a dimension's lower bound and length is greater than Int32.MaxValue.

Examples

The following code example shows how to create and initialize a multidimensional Array with specified lower bounds.

C#

```
using System;
public class SamplesArray4  {

   public static void Main()  {

      // Creates and initializes a multidimensional Array of type string.
      int[] myLengthsArray = new int[2] { 3, 5 };
      int[] myBoundsArray = new int[2] { 2, 3 };
      Array myArray=Array.CreateInstance( typeof(string), myLengthsArray,
myBoundsArray );
      for ( int i = myArray.GetLowerBound(0); i <=
myArray.GetUpperBound(0); i++ )
         for ( int j = myArray.GetLowerBound(1); j <=
myArray.GetUpperBound(1); j++ )  {
            int[] myIndicesArray = new int[2] { i, j };
            myArray.SetValue( Convert.ToString(i) + j, myIndicesArray );
         }
```

```
      // Displays the lower bounds and the upper bounds of each
dimension.
      Console.WriteLine( "Bounds:\tLower\tUpper" );
      for ( int i = 0; i < myArray.Rank; i++ )
         Console.WriteLine( "{0}:\t{1}\t{2}", i,
myArray.GetLowerBound(i), myArray.GetUpperBound(i) );

      // Displays the values of the Array.
      Console.WriteLine( "The Array contains the following values:" );
      PrintValues( myArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
         } else  {
            Console.WriteLine();
            i = 1;
         }
         Console.Write( "\t{0}", myEnumerator.Current );
      }
      Console.WriteLine();
   }
}
/*
This code produces the following output.

Bounds:    Lower    Upper
0:    2    4
1:    3    7
The Array contains the following values:
    23    24    25    26    27
    33    34    35    36    37
    43    44    45    46    47
*/
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

The `lengths` and `lowerBounds` arrays must have the same number of elements. The number of elements in the `lengths` array must equal the number of dimensions in the new Array.

Each element of the `lengths` array must specify the length of the corresponding dimension in the new Array.

Each element of the `lowerBounds` array must specify the lower bound of the corresponding dimension in the new Array. Generally, the .NET class library and many programming languages do not handle non-zero lower bounds.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is the product of all values in `lengths`.

> **Note**
>
> Not all languages support arrays with non-zero lower bounds, and therefore you may not be able to cast instances of non-zero based **Array** to the language's array type. For example, you cannot cast a 1-dimensional integer array with the lower bound 6 to C#'s `int[]` type. This results in **InvalidCastException** during runtime with the message "Unable to cast object of type 'System.Int32[*]' to type 'System.Int32[]'.", where the asterisk (*) means non-zero based index. However, you can cast zero based arrays of any rank created with **CreateInstance(Type, Int32[], Int32[])** to the language's array. For example, you can cast a 2-dimensional zero based integer array created with this method to C#'s `int[,]` type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CreateInstance(Type, Int32, Int32, Int32)

Creates a three-dimensional Array of the specified Type and dimension lengths, with zero-based indexing.

```
C#

public static Array CreateInstance (Type elementType, int length1, int
length2, int length3);
```

Parameters

elementType Type

The Type of the Array to create.

length1 Int32

The size of the first dimension of the Array to create.

length2 Int32

The size of the second dimension of the Array to create.

length3 Int32

The size of the third dimension of the Array to create.

Returns

Array

A new three-dimensional Array of the specified Type with the specified length for each dimension, using zero-based indexing.

Exceptions

ArgumentNullException

`elementType` is `null`.

ArgumentException

`elementType` is not a valid Type.

NotSupportedException

`elementType` is not supported. For example, Void is not supported.

-or-

`elementType` is an open generic type.

ArgumentOutOfRangeException

`length1` is less than zero.

-or-

`length2` is less than zero.

-or-

`length3` is less than zero.

Examples

The following code example shows how to create and initialize a three-dimensional Array.

C#

```
using System;
public class SamplesArray2  {

   public static void Main()  {

      // Creates and initializes a three-dimensional Array of type
Object.
      Array my3DArray=Array.CreateInstance( typeof(Object), 2, 3, 4 );
      for ( int i = my3DArray.GetLowerBound(0); i <=
my3DArray.GetUpperBound(0); i++ )
         for ( int j = my3DArray.GetLowerBound(1); j <=
my3DArray.GetUpperBound(1); j++ )
            for ( int k = my3DArray.GetLowerBound(2); k <=
my3DArray.GetUpperBound(2); k++ )
               my3DArray.SetValue( "abc" + i + j + k, i, j, k );

      // Displays the values of the Array.
      Console.WriteLine( "The three-dimensional Array contains the
following values:" );
      PrintValues( my3DArray );
   }

   public static void PrintValues( Array myArr )  {
      System.Collections.IEnumerator myEnumerator =
myArr.GetEnumerator();
      int i = 0;
      int cols = myArr.GetLength( myArr.Rank - 1 );
      while ( myEnumerator.MoveNext() )  {
         if ( i < cols )  {
            i++;
```

```
            } else  {
               Console.WriteLine();
               i = 1;
            }
            Console.Write( "\t{0}", myEnumerator.Current );
         }
         Console.WriteLine();
      }
   }
   /*
   This code produces the following output.

   The three-dimensional Array contains the following values:
       abc000    abc001    abc002    abc003
       abc010    abc011    abc012    abc013
       abc020    abc021    abc022    abc023
       abc100    abc101    abc102    abc103
       abc110    abc111    abc112    abc113
       abc120    abc121    abc122    abc123
   */
```

Remarks

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

Reference-type elements are initialized to `null`. Value-type elements are initialized to zero.

This method is an O(`n`) operation, where `n` is the product of `length1`, `length2`, and `length3`.

In F#, the Array3D.zeroCreate function can be used instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.Empty<T> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an empty array.

C#

```
public static T[] Empty<T> ();
```

Type Parameters

T

The type of the elements of the array.

Returns

T[]

An empty array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.Exists<T>(T[], Predicate<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified array contains elements that match the conditions defined by the specified predicate.

```
C#

public static bool Exists<T> (T[] array, Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the elements to search for.

Returns

Boolean

`true` if `array` contains one or more elements that match the conditions defined by the specified predicate; otherwise, `false`.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Examples

The following example specifies the match conditions for the Exists method using lambda expressions to check whether a planet starts with a given letter or whether the planet is found on the given array.

C#

```
using System;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] planets = { "Mercury", "Venus",
                "Earth", "Mars", "Jupiter",
                "Saturn", "Uranus", "Neptune" };

            Console.WriteLine("One or more planets begin with 'M': {0}",
                Array.Exists(planets, element => element.StartsWith("M")));

            Console.WriteLine("One or more planets begin with 'T': {0}",
                Array.Exists(planets, element => element.StartsWith("T")));

            Console.WriteLine("Is Pluto one of the planets? {0}",
                Array.Exists(planets, element => element == "Pluto"));
        }
    }
}
// The example displays the following output:
//       One or more planets begin with 'M': True
//       One or more planets begin with 'T': False
//       Is Pluto one of the planets? False
```

The following example uses the Exists method to indicate whether any names in a string array begin with a specified character. The example instantiates a `StringSearcher` object by passing the string to search for to its class constructor. The `StringSearcher.StartsWith` method has same signature as the Predicate<T> delegate. When the Exists method is called, each member of the array is passed to the delegate until it returns `true` or iterates all the elements in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] names = { "Adam", "Adel", "Bridgette", "Carla",
                         "Charles", "Daniel", "Elaine", "Frances",
                         "George", "Gillian", "Henry", "Irving",
                         "James", "Janae", "Lawrence", "Miguel",
                         "Nicole", "Oliver", "Paula", "Robert",
                         "Stephen", "Thomas", "Vanessa",
                         "Veronica", "Wilberforce" };
      Char[] charsToFind = { 'A', 'K', 'W', 'Z' };

      foreach (var charToFind in charsToFind)
         Console.WriteLine("One or more names begin with '{0}': {1}",
                           charToFind,
                           Array.Exists(names, (new
StringSearcher(charToFind)).StartsWith));
   }
}

public class StringSearcher
{
   char firstChar;

   public StringSearcher(char firstChar)
   {
      this.firstChar = char.ToUpper(firstChar);
   }

   public bool StartsWith(string s)
   {
      if (string.IsNullOrEmpty(s)) return false;

      if(s.Substring(0, 1).ToUpper() == firstChar.ToString())
         return true;
      else
         return false;
   }
}
// The example displays the following output:
//       One or more names begin with 'A': True
//       One or more names begin with 'K': False
//       One or more names begin with 'W': True
//       One or more names begin with 'Z': False
```

You can also use a lambda expression rather than explicitly define a method whose signature corresponds to that of the delegate. The following example replaces the

`StringSearcher` class and its `StartsWith` method with a lambda expression.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] names = { "Adam", "Adel", "Bridgette", "Carla",
                         "Charles", "Daniel", "Elaine", "Frances",
                         "George", "Gillian", "Henry", "Irving",
                         "James", "Janae", "Lawrence", "Miguel",
                         "Nicole", "Oliver", "Paula", "Robert",
                         "Stephen", "Thomas", "Vanessa",
                         "Veronica", "Wilberforce" };
      Char[] charsToFind = { 'A', 'K', 'W', 'Z' };

      foreach (var charToFind in charsToFind)
         Console.WriteLine("One or more names begin with '{0}': {1}",
                           charToFind,
                           Array.Exists(names,
                                        s => { if (string.IsNullOrEmpty(s))
                                                   return false;

                                               if (s.Substring(0,
1).ToUpper() == charToFind.ToString())
                                                   return true;
                                               else
                                                   return false;
                                             } ));
   }
}
// The example displays the following output:
//       One or more names begin with 'A': True
//       One or more names begin with 'K': False
//       One or more names begin with 'W': True
//       One or more names begin with 'Z': False
```

Remarks

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>, and processing is stopped when a match is found.

ⓘ **Note**

In C# and Visual Basic, it is not necessary to create the **Predicate<T>** delegate explicitly. These languages infer the correct delegate from context and create it automatically. In F#, functions and lambda expressions are implicitly converted.

This method is an O(`n`) operation, where `n` is the Length of `array`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- TrueForAll<T>(T[], Predicate<T>)
- Predicate<T>
- Lambda Expressions (C# Programming Guide)
- Lambda Expressions (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Fill Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Fill<T>(T[], T)	Assigns the given `value` of type `T` to each element of the specified `array`.
Fill<T>(T[], T, Int32, Int32)	Assigns the given `value` of type `T` to the elements of the specified `array` which are within the range of `startIndex` (inclusive) and the next `count` number of indices.

Fill<T>(T[], T)

Assigns the given `value` of type `T` to each element of the specified `array`.

C#

```
public static void Fill<T> (T[] array, T value);
```

Type Parameters

T

The type of the elements in the array.

Parameters

array T[]

The array to be filled.

value T

The value to assign to each array element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Fill<T>(T[], T, Int32, Int32)

Assigns the given `value` of type `T` to the elements of the specified `array` which are within the range of `startIndex` (inclusive) and the next `count` number of indices.

```
C#

public static void Fill<T> (T[] array, T value, int startIndex, int
count);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The Array to be filled.

`value` T

The new value for the elements in the specified range.

`startIndex` Int32

A 32-bit integer that represents the index in the Array at which filling begins.

`count` Int32

The number of elements to copy.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Find<T>(T[], Predicate<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for an element that matches the conditions defined by the specified predicate, and returns the first occurrence within the entire Array.

```
C#

public static T? Find<T> (T[] array, Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based array to search.

`match` Predicate<T>

The predicate that defines the conditions of the element to search for.

Returns

T

The first element that matches the conditions defined by the specified predicate, if found; otherwise, the default value for type `T`.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Examples

The following example uses a Predicate<T> delegate with the Find generic method to search an array of Point structures. The method the delegate represents, `ProductGT10`, returns `true` if the product of the X and Y fields is greater than 100,000. The Find method calls the delegate for each element of the array, returning the first point that meets the test condition.

> **Note**
>
> Visual Basic, C#, and F# users do not have to create the delegate explicitly or specify the type argument of the generic method. The compilers determine the necessary types from the method arguments you supply.

C#

```
using System;
using System.Drawing;

public class Example
{
    public static void Main()
    {
        // Create an array of five Point structures.
        Point[] points = { new Point(100, 200),
            new Point(150, 250), new Point(250, 375),
            new Point(275, 395), new Point(295, 450) };

        // Find the first Point structure for which X times Y
        // is greater than 100000.
        Point first = Array.Find(points, ProductGT10);

        // Display the first structure found.
        Console.WriteLine("Found: X = {0}, Y = {1}", first.X, first.Y);
    }

    // Return true if X times Y is greater than 100000.
    private static bool ProductGT10(Point p)
    {
        return p.X * p.Y > 100000;
    }
```

```
}
// The example displays the following output:
//       Found: X = 275, Y = 395
```

Rather than explicitly defining a method with the necessary signature, instantiating a Predicate<T> delegate, and passing the delegate to the Find method, it is customary to use a lambda expression. The following example is identical to the previous one, except that it uses a lambda expression as the `match` argument.

C#

```
using System;
using System.Drawing;

public class Example
{
    public static void Main()
    {
        // Create an array of five Point structures.
        Point[] points = { new Point(100, 200),
            new Point(150, 250), new Point(250, 375),
            new Point(275, 395), new Point(295, 450) };

        // Find the first Point structure for which X times Y
        // is greater than 100000.
        Point first = Array.Find(points, p => p.X * p.Y > 100000);

        // Display the first structure found.
        Console.WriteLine("Found: X = {0}, Y = {1}", first.X, first.Y);
    }
}
// The example displays the following output:
//       Found: X = 275, Y = 395
```

Remarks

The Predicate<T> is a delegate to a method or a lambda expression that returns `true` if the object passed to it matches the conditions defined in the delegate or lambda expression. The elements of `array` are individually passed to the Predicate<T>, starting with the first element and ending with the last element. Processing is stopped when a match is found.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In F#, the Array.find function can be used instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exists<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.FindAll<T>(T[], Predicate<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves all the elements that match the conditions defined by the specified predicate.

C#

```
public static T[] FindAll<T> (T[] array, Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the elements to search for.

Returns

T[]

An Array containing all the elements that match the conditions defined by the specified predicate, if found; otherwise, an empty Array.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Examples

The following example creates an array of 50 random numbers with values that can range from 0 to 1,000. It then calls the FindAll method with a lambda expression that returns the values that range from 300 to 600. Note that the lambda expression is passed a parameter named `x`; this represents the individual array member that is passed to the Predicate<T>. Also note that the local `lBound` and `uBound` variables are accessible within the lambda expression.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      // Get an array of n random integers.
      int[] values = GetArray(50, 0, 1000);
      int lBound = 300;
      int uBound = 600;
      int[] matchedItems = Array.FindAll(values, x =>
                                         x >= lBound && x <= uBound);
      for (int ctr = 0; ctr < matchedItems.Length; ctr++) {
         Console.Write("{0}  ", matchedItems[ctr]);
         if ((ctr + 1) % 12 == 0)
            Console.WriteLine();
      }
   }

   private static int[] GetArray(int n, int lower, int upper)
   {
      Random rnd = new Random();
      List<int> list = new List<int>();
      for (int ctr = 1; ctr <= n; ctr++)
         list.Add(rnd.Next(lower, upper + 1));

      return list.ToArray();
   }
}
// The example displays output similar to the following:
//       542  398  356  351  348  301  562  599  575  400  569  306
//       535  416  393  385
```

The following code example demonstrates the Find, FindLast, and FindAll generic methods. An array of strings is created, containing 8 dinosaur names, two of which (at positions 1 and 5) end with "saurus". The code example also defines a search predicate method named `EndsWithSaurus`, which accepts a string parameter and returns a Boolean value indicating whether the input string ends in "saurus".

The Find generic method traverses the array from the beginning, passing each element in turn to the `EndsWithSaurus` method. The search stops when the `EndsWithSaurus` method returns `true` for the element "Amargasaurus".

> **Note**
>
> In C#, F#, and Visual Basic, it is not necessary to create the `Predicate<string>` delegate (`Predicate(Of String)` in Visual Basic) explicitly. These languages infer the correct delegate from context and create it automatically.

The FindLast generic method is used to search the array backward from the end. It finds the element "Dilophosaurus" at position 5. The FindAll generic method is used to return an array containing all the elements that end in "saurus". The elements are displayed.

The code example also demonstrates the Exists and TrueForAll generic methods.

C#

```
using System;

public class DinoDiscoverySet
{
    public static void Main()
    {
        string[] dinosaurs =
        {
            "Compsognathus", "Amargasaurus", "Oviraptor",
            "Velociraptor",  "Deinonychus",  "Dilophosaurus",
            "Gallimimus",    "Triceratops"
        };

        DinoDiscoverySet GoMesozoic = new DinoDiscoverySet(dinosaurs);

        GoMesozoic.DiscoverAll();
        GoMesozoic.DiscoverByEnding("saurus");
    }

    private string[] dinosaurs;

    public DinoDiscoverySet(string[] items)
    {
```

```
        dinosaurs = items;
    }

    public void DiscoverAll()
    {
        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }
    }

    public void DiscoverByEnding(string Ending)
    {
        Predicate<string> dinoType;

        switch (Ending.ToLower())
        {
            case "raptor":
                dinoType = EndsWithRaptor;
                break;
            case "tops":
                dinoType = EndsWithTops;
                break;
            case "saurus":
            default:
                dinoType = EndsWithSaurus;
                break;
        }
        Console.WriteLine(
            "\nArray.Exists(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.Exists(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.TrueForAll(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.TrueForAll(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.Find(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.Find(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.FindLast(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.FindLast(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.FindAll(dinosaurs, \"{0}\"):", Ending);

        string[] subArray =
            Array.FindAll(dinosaurs, dinoType);
```

```
        foreach(string dinosaur in subArray)
        {
            Console.WriteLine(dinosaur);
        }
    }

    // Search predicate returns true if a string ends in "saurus".
    private bool EndsWithSaurus(string s)
    {
        if ((s.Length > 5) &&
            (s.Substring(s.Length - 6).ToLower() == "saurus"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Search predicate returns true if a string ends in "raptor".
    private bool EndsWithRaptor(String s)
    {
        if ((s.Length > 5) &&
            (s.Substring(s.Length - 6).ToLower() == "raptor"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Search predicate returns true if a string ends in "tops".
    private bool EndsWithTops(String s)
    {
        if ((s.Length > 3) &&
            (s.Substring(s.Length - 4).ToLower() == "tops"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

/* This code example produces the following output:

Compsognathus
Amargasaurus
Oviraptor
Velociraptor
```

```
Deinonychus
Dilophosaurus
Gallimimus
Triceratops

Array.Exists(dinosaurs, "saurus"): True

Array.TrueForAll(dinosaurs, "saurus"): False

Array.Find(dinosaurs, "saurus"): Amargasaurus

Array.FindLast(dinosaurs, "saurus"): Dilophosaurus

Array.FindAll(dinosaurs, "saurus"):
Amargasaurus
Dilophosaurus
*/
```

Remarks

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>, and the elements that match the conditions are saved in the returned array.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In F#, the Array.filter function may be used instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)

- FindLast<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.FindIndex Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the first occurrence within an Array or a portion of it.

Overloads

Expand table

FindIndex<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the entire Array.
FindIndex<T>(T[], Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that extends from the specified index to the last element.
FindIndex<T>(T[], Int32, Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that starts at the specified index and contains the specified number of elements.

Examples

The following code example demonstrates all three overloads of the FindIndex generic method. An array of strings is created, containing 8 dinosaur names, two of which (at positions 1 and 5) end with "saurus". The code example also defines a search predicate method named `EndsWithSaurus`, which accepts a string parameter and returns a Boolean value indicating whether the input string ends in "saurus".

The FindIndex<T>(T[], Predicate<T>) method overload traverses the array from the beginning, passing each element in turn to the `EndsWithSaurus` method. The search stops when the `EndsWithSaurus` method returns `true` for the element at position 1.

> **Note**
>
> In C#, F#, and Visual Basic, it is not necessary to create the `Predicate<string>` delegate (`Predicate(Of String)` in Visual Basic) explicitly. These languages infer the correct delegate from context and create it automatically.

The FindIndex<T>(T[], Int32, Predicate<T>) method overload is used to search the array beginning at position 2 and continuing to the end of the array. It finds the element at position 5. Finally, the FindIndex<T>(T[], Int32, Int32, Predicate<T>) method overload is used to search the range of three elements beginning at position 2. It returns -1 because there are no dinosaur names in that range that end with "saurus".

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = { "Compsognathus",
            "Amargasaurus",   "Oviraptor",      "Velociraptor",
            "Deinonychus",    "Dilophosaurus",  "Gallimimus",
            "Triceratops" };

        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine(
            "\nArray.FindIndex(dinosaurs, EndsWithSaurus): {0}",
            Array.FindIndex(dinosaurs, EndsWithSaurus));

        Console.WriteLine(
            "\nArray.FindIndex(dinosaurs, 2, EndsWithSaurus): {0}",
            Array.FindIndex(dinosaurs, 2, EndsWithSaurus));

        Console.WriteLine(
            "\nArray.FindIndex(dinosaurs, 2, 3, EndsWithSaurus): {0}",
            Array.FindIndex(dinosaurs, 2, 3, EndsWithSaurus));
    }

    // Search predicate returns true if a string ends in "saurus".
    private static bool EndsWithSaurus(String s)
    {
        if ((s.Length > 5) &&
            (s.Substring(s.Length - 6).ToLower() == "saurus"))
        {
```

```
            return true;
        }
        else
        {
            return false;
        }
    }
}

/* This code example produces the following output:

Compsognathus
Amargasaurus
Oviraptor
Velociraptor
Deinonychus
Dilophosaurus
Gallimimus
Triceratops

Array.FindIndex(dinosaurs, EndsWithSaurus): 1

Array.FindIndex(dinosaurs, 2, EndsWithSaurus): 5

Array.FindIndex(dinosaurs, 2, 3, EndsWithSaurus): -1
 */
```

FindIndex<T>(T[], Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the entire Array.

C#

```
public static int FindIndex<T> (T[] array, Predicate<T> match);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based Array to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the first occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Remarks

The Array is searched forward starting at the first element and ending at the last element.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is the Length of `array`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FindIndex<T>(T[], Int32, Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that extends from the specified index to the last element.

C#

```
public static int FindIndex<T> (T[] array, int startIndex, Predicate<T>
match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`startIndex` Int32

The zero-based starting index of the search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the first occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

Remarks

The Array is searched forward starting at `startIndex` and ending at the last element.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is the number of elements from `startIndex` to the end of `array`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FindIndex<T>(T[], Int32, Int32, Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the range of elements in the Array that starts at the specified index and contains the specified number of elements.

C#

```
public static int FindIndex<T> (T[] array, int startIndex, int count,
Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`startIndex` Int32

The zero-based starting index of the search.

`count` Int32

The number of elements in the section to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the first occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

Remarks

The Array is searched forward starting at `startIndex` and ending at `startIndex` plus `count` minus 1, if `count` is greater than 0.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is `count`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)

- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.FindLast<T>(T[], Predicate<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for an element that matches the conditions defined by the specified predicate, and returns the last occurrence within the entire Array.

```
C#

public static T? FindLast<T> (T[] array, Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

T

The last element that matches the conditions defined by the specified predicate, if found; otherwise, the default value for type `T`.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Examples

The following code example demonstrates the Find, FindLast, and FindAll generic methods. An array of strings is created, containing 8 dinosaur names, two of which (at positions 1 and 5) end with "saurus". The code example also defines a search predicate method named `EndsWithSaurus`, which accepts a string parameter and returns a Boolean value indicating whether the input string ends in "saurus".

The Find generic method traverses the array from the beginning, passing each element in turn to the `EndsWithSaurus` method. The search stops when the `EndsWithSaurus` method returns `true` for the element "Amargasaurus".

> **Note**
>
> In C# and Visual Basic, it is not necessary to create the `Predicate<string>` delegate (`Predicate(Of String)` in Visual Basic) explicitly. These languages infer the correct delegate from context and create it automatically.

The FindLast generic method is used to search the array backward from the end. It finds the element "Dilophosaurus" at position 5. The FindAll generic method is used to return an array containing all the elements that end in "saurus". The elements are displayed.

The code example also demonstrates the Exists and TrueForAll generic methods.

C#

```
using System;

public class DinoDiscoverySet
{
    public static void Main()
    {
        string[] dinosaurs =
        {
            "Compsognathus", "Amargasaurus", "Oviraptor",
            "Velociraptor",  "Deinonychus",  "Dilophosaurus",
            "Gallimimus",    "Triceratops"
        };
```

```
        DinoDiscoverySet GoMesozoic = new DinoDiscoverySet(dinosaurs);

        GoMesozoic.DiscoverAll();
        GoMesozoic.DiscoverByEnding("saurus");
    }

    private string[] dinosaurs;

    public DinoDiscoverySet(string[] items)
    {
        dinosaurs = items;
    }

    public void DiscoverAll()
    {
        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }
    }

    public void DiscoverByEnding(string Ending)
    {
        Predicate<string> dinoType;

        switch (Ending.ToLower())
        {
            case "raptor":
                dinoType = EndsWithRaptor;
                break;
            case "tops":
                dinoType = EndsWithTops;
                break;
            case "saurus":
            default:
                dinoType = EndsWithSaurus;
                break;
        }
        Console.WriteLine(
            "\nArray.Exists(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.Exists(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.TrueForAll(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.TrueForAll(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.Find(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.Find(dinosaurs, dinoType));

        Console.WriteLine(
```

```
            "\nArray.FindLast(dinosaurs, \"{0}\"): {1}",
            Ending,
            Array.FindLast(dinosaurs, dinoType));

        Console.WriteLine(
            "\nArray.FindAll(dinosaurs, \"{0}\"):", Ending);

        string[] subArray =
            Array.FindAll(dinosaurs, dinoType);

        foreach(string dinosaur in subArray)
        {
            Console.WriteLine(dinosaur);
        }
    }

    // Search predicate returns true if a string ends in "saurus".
    private bool EndsWithSaurus(string s)
    {
        if ((s.Length > 5) &&
            (s.Substring(s.Length - 6).ToLower() == "saurus"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Search predicate returns true if a string ends in "raptor".
    private bool EndsWithRaptor(String s)
    {
        if ((s.Length > 5) &&
            (s.Substring(s.Length - 6).ToLower() == "raptor"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Search predicate returns true if a string ends in "tops".
    private bool EndsWithTops(String s)
    {
        if ((s.Length > 3) &&
            (s.Substring(s.Length - 4).ToLower() == "tops"))
        {
            return true;
        }
        else
        {
            return false;
```

```
        }
    }
}

/* This code example produces the following output:

Compsognathus
Amargasaurus
Oviraptor
Velociraptor
Deinonychus
Dilophosaurus
Gallimimus
Triceratops

Array.Exists(dinosaurs, "saurus"): True

Array.TrueForAll(dinosaurs, "saurus"): False

Array.Find(dinosaurs, "saurus"): Amargasaurus

Array.FindLast(dinosaurs, "saurus"): Dilophosaurus

Array.FindAll(dinosaurs, "saurus"):
Amargasaurus
Dilophosaurus
*/
```

Remarks

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>, moving backward in the Array, starting with the last element and ending with the first element. Processing is stopped when a match is found.

This method is an O(`n`) operation, where `n` is the Length of `array`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.FindLastIndex Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the last occurrence within an Array or a portion of it.

Overloads

Expand table

FindLastIndex<T>(T[], Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the entire Array.
FindLastIndex<T>(T[], Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.
FindLastIndex<T>(T[], Int32, Int32, Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.

Examples

The following code example demonstrates all three overloads of the FindLastIndex generic method. An array of strings is created, containing 8 dinosaur names, two of which (at positions 1 and 5) end with "saurus". The code example also defines a search predicate method named `EndsWithSaurus`, which accepts a string parameter and returns a Boolean value indicating whether the input string ends in "saurus".

The FindLastIndex<T>(T[], Predicate<T>) method overload traverses the array backward from the end, passing each element in turn to the `EndsWithSaurus` method. The search stops when the `EndsWithSaurus` method returns `true` for the element at position 5.

> **Note**
>
> In C#, F# and Visual Basic, it is not necessary to create the `Predicate<string>` delegate (`Predicate(Of String)` in Visual Basic) explicitly. These languages infer the correct delegate from context and create it automatically.

The FindLastIndex<T>(T[], Int32, Predicate<T>) method overload is used to search the array beginning at position 4 and continuing backward to the beginning of the array. It finds the element at position 1. Finally, the FindLastIndex<T>(T[], Int32, Int32, Predicate<T>) method overload is used to search the range of three elements beginning at position 4 and working backward (that is, elements 4, 3, and 2). It returns -1 because there are no dinosaur names in that range that end with "saurus".

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = { "Compsognathus",
            "Amargasaurus",   "Oviraptor",      "Velociraptor",
            "Deinonychus",    "Dilophosaurus",  "Gallimimus",
            "Triceratops" };

        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine(
            "\nArray.FindLastIndex(dinosaurs, EndsWithSaurus): {0}",
            Array.FindLastIndex(dinosaurs, EndsWithSaurus));

        Console.WriteLine(
            "\nArray.FindLastIndex(dinosaurs, 4, EndsWithSaurus): {0}",
            Array.FindLastIndex(dinosaurs, 4, EndsWithSaurus));

        Console.WriteLine(
            "\nArray.FindLastIndex(dinosaurs, 4, 3, EndsWithSaurus): {0}",
            Array.FindLastIndex(dinosaurs, 4, 3, EndsWithSaurus));
    }

    // Search predicate returns true if a string ends in "saurus".
    private static bool EndsWithSaurus(String s)
    {
        if ((s.Length > 5) &&
```

```
            (s.Substring(s.Length - 6).ToLower() == "saurus"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

/* This code example produces the following output:

Compsognathus
Amargasaurus
Oviraptor
Velociraptor
Deinonychus
Dilophosaurus
Gallimimus
Triceratops

Array.FindLastIndex(dinosaurs, EndsWithSaurus): 5

Array.FindLastIndex(dinosaurs, 4, EndsWithSaurus): 1

Array.FindLastIndex(dinosaurs, 4, 3, EndsWithSaurus): -1
 */
```

FindLastIndex<T>(T[], Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the entire Array.

C#

```
public static int FindLastIndex<T> (T[] array, Predicate<T> match);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based Array to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the last occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Remarks

The Array is searched backward starting at the last element and ending at the first element.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is the Length of `array`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FindLastIndex<T>(T[], Int32, Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.

C#

```
public static int FindLastIndex<T> (T[] array, int startIndex,
Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`startIndex` Int32

The zero-based starting index of the backward search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the last occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

Remarks

The Array is searched backward starting at `startIndex` and ending at the first element.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is the number of elements from the beginning of `array` to `startIndex`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FindLastIndex<T>(T[], Int32, Int32, Predicate<T>)

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.

C#

```
public static int FindLastIndex<T> (T[] array, int startIndex, int count,
Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`startIndex` Int32

The zero-based starting index of the backward search.

`count` Int32

The number of elements in the section to search.

`match` Predicate<T>

The Predicate<T> that defines the conditions of the element to search for.

Returns

Int32

The zero-based index of the last occurrence of an element that matches the conditions defined by `match`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

Remarks

The Array is searched backward starting at `startIndex` and ending at `startIndex` minus `count` plus 1, if `count` is greater than 0.

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>.

This method is an O(`n`) operation, where `n` is `count`.

See also

- Exists<T>(T[], Predicate<T>)
- Find<T>(T[], Predicate<T>)
- FindLast<T>(T[], Predicate<T>)
- FindAll<T>(T[], Predicate<T>)
- BinarySearch
- IndexOf
- LastIndexOf
- Predicate<T>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.ForEach<T>(T[], Action<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs the specified action on each element of the specified array.

```
C#
public static void ForEach<T> (T[] array, Action<T> action);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based Array on whose elements the action is to be performed.

action Action<T>

The Action<T> to perform on each element of `array`.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`action` is `null`.

Examples

The following example shows how to use ForEach to display the squares of each element in an integer array.

C#

```
using System;

public class SamplesArray
{
    public static void Main()
    {
        // create a three element array of integers
        int[] intArray = new int[] {2, 3, 4};

        // set a delegate for the ShowSquares method
        Action<int> action = new Action<int>(ShowSquares);

        Array.ForEach(intArray, action);
    }

    private static void ShowSquares(int val)
    {
        Console.WriteLine("{0:d} squared = {1:d}", val, val*val);
    }
}

/*
This code produces the following output:

2 squared = 4
3 squared = 9
4 squared = 16
*/
```

Remarks

The Action<T> is a delegate to a method that performs an action on the object passed to it. The elements of `array` are individually passed to the Action<T>.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In F#, the Array.iter function can be used instead.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Action<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an IEnumerator for the Array.

C#

```
public System.Collections.IEnumerator GetEnumerator ();
```

Returns

IEnumerator

An IEnumerator for the Array.

Implements

GetEnumerator()

Examples

The following code example shows how to use GetEnumerator to list the elements of an array.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Creates and initializes a new Array.
      String[] myArr = new String[10];
      myArr[0] = "The";
      myArr[1] = "quick";
      myArr[2] = "brown";
      myArr[3] = "fox";
```

```
        myArr[4] = "jumps";
        myArr[5] = "over";
        myArr[6] = "the";
        myArr[7] = "lazy";
        myArr[8] = "dog";

        // Displays the values of the Array.
        int i = 0;
        System.Collections.IEnumerator myEnumerator = myArr.GetEnumerator();
        Console.WriteLine( "The Array contains the following values:" );
        while (( myEnumerator.MoveNext() ) && ( myEnumerator.Current != null
))
           Console.WriteLine( "[{0}] {1}", i++, myEnumerator.Current );
    }
}


/*
This code produces the following output.

The Array contains the following values:
[0] The
[1] quick
[2] brown
[3] fox
[4] jumps
[5] over
[6] the
[7] lazy
[8] dog

*/
```

Remarks

The `foreach` statement of the C# language (`for each` in C++, `For Each` in Visual Basic) hides the complexity of the enumerators. Therefore, using `foreach` is recommended, instead of directly manipulating the enumerator.

Enumerators can be used to read the data in the collection, but they cannot be used to modify the underlying collection.

Initially, the enumerator is positioned before the first element in the collection. Reset also brings the enumerator back to this position. At this position, Current is undefined. Therefore, you must call MoveNext to advance the enumerator to the first element of the collection before reading the value of Current.

Current returns the same object until either MoveNext or Reset is called. MoveNext sets Current to the next element.

If MoveNext passes the end of the collection, the enumerator is positioned after the last element in the collection and MoveNext returns `false`. When the enumerator is at this position, subsequent calls to MoveNext also return `false`. If the last call to MoveNext returned `false`, Current is undefined. To set Current to the first element of the collection again, you can call Reset followed by MoveNext.

An enumerator remains valid as long as the collection remains unchanged. If changes are made to the collection, such as adding, modifying, or deleting elements, the enumerator is irrecoverably invalidated and its behavior is undefined.

The enumerator does not have exclusive access to the collection; therefore, enumerating through a collection is intrinsically not a thread safe procedure. To guarantee thread safety during enumeration, you can lock the collection during the entire enumeration. To allow the collection to be accessed by multiple threads for reading and writing, you must implement your own synchronization.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.GetLength(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.

C#

```
public int GetLength (int dimension);
```

Parameters

`dimension` Int32

A zero-based dimension of the Array whose length needs to be determined.

Returns

Int32

A 32-bit integer that represents the number of elements in the specified dimension.

Exceptions

IndexOutOfRangeException

`dimension` is less than zero.

-or-

`dimension` is equal to or greater than Rank.

Examples

The following example shows how to use GetLength to display the dimensions of two arrays with different ranks.

C#

```
using System;

public class SamplesArray
{
    public static void Main()
    {
        // make a single dimension array
        Array MyArray1 = Array.CreateInstance(typeof(int), 5);

        // make a 3 dimensional array
        Array MyArray2 = Array.CreateInstance(typeof(int), 5, 3, 2);

        // make an array container
        Array BossArray = Array.CreateInstance(typeof(Array), 2);
        BossArray.SetValue(MyArray1, 0);
        BossArray.SetValue(MyArray2, 1);

        int i = 0, j, rank;
        foreach (Array anArray in BossArray)
        {
            rank = anArray.Rank;
            if (rank > 1)
            {
                Console.WriteLine("Lengths of {0:d} dimension array[{1:d}]",
rank, i);
                // show the lengths of each dimension
                for (j = 0; j < rank; j++)
                {
                    Console.WriteLine("    Length of dimension({0:d}) =
{1:d}", j, anArray.GetLength(j));
                }
            }
            else
            {
                Console.WriteLine("Lengths of single dimension
array[{0:d}]", i);
            }
            // show the total length of the entire array or all dimensions
            Console.WriteLine("    Total length of the array = {0:d}",
anArray.Length);
            Console.WriteLine();
            i++;
        }
    }
}

/*
This code produces the following output:

Lengths of single dimension array[0]
    Total length of the array = 5

Lengths of 3 dimension array[1]
    Length of dimension(0) = 5
```

```
    Length of dimension(1) = 3
    Length of dimension(2) = 2
    Total length of the array = 30
*/
```

Remarks

An example of GetLength is `GetLength(0)`, which returns the number of elements in the first dimension of the Array.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Length
- GetLongLength(Int32)
- Count
- Rank

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.GetLongLength(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.

```
C#
public long GetLongLength (int dimension);
```

Parameters

dimension Int32

A zero-based dimension of the Array whose length needs to be determined.

Returns

Int64

A 64-bit integer that represents the number of elements in the specified dimension.

Exceptions

IndexOutOfRangeException

`dimension` is less than zero.

-or-

`dimension` is equal to or greater than Rank.

Remarks

An example of GetLongLength is `GetLongLength(0)`, which returns the number of elements in the first dimension of the Array.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Length
- GetLength(Int32)
- Count
- Rank

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.GetLowerBound(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the index of the first element of the specified dimension in the array.

```
C#
public int GetLowerBound (int dimension);
```

Parameters

`dimension` Int32

A zero-based dimension of the array whose starting index needs to be determined.

Returns

Int32

The index of the first element of the specified dimension in the array.

Exceptions

IndexOutOfRangeException

`dimension` is less than zero.

-or-

`dimension` is equal to or greater than Rank.

Examples

The following example uses the GetLowerBound and GetUpperBound methods to display the bounds of a one-dimensional and two-dimensional array and to display the values of their array elements.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      // Create a one-dimensional integer array.
      int[] integers = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
      // Get the upper and lower bound of the array.
      int upper = integers.GetUpperBound(0);
      int lower = integers.GetLowerBound(0);
      Console.WriteLine($"Elements from index {lower} to {upper}:");
      // Iterate the array.
      for (int ctr = lower; ctr <= upper; ctr++)
        Console.Write($"{(ctr == lower ?"    " : "")}{integers[ctr]}" +
                      $"{(ctr < upper ? ", " : Environment.NewLine)}");

      Console.WriteLine();

      // Create a two-dimensional integer array.
      int[,] integers2d= { {2, 4}, {3, 9}, {4, 16}, {5, 25},
                           {6, 36}, {7, 49}, {8, 64}, {9, 81} };
      // Get the number of dimensions.
      int rank = integers2d.Rank;
      Console.WriteLine($"Number of dimensions: {rank}");
      for (int ctr = 0; ctr < rank; ctr++)
        Console.WriteLine($"   Dimension {ctr}: " +
                          $"from {integers2d.GetLowerBound(ctr)} to
{integers2d.GetUpperBound(ctr)}");

      // Iterate the 2-dimensional array and display its values.
      Console.WriteLine("   Values of array elements:");
      for (int outer = integers2d.GetLowerBound(0); outer <=
integers2d.GetUpperBound(0);
           outer++)
        for (int inner = integers2d.GetLowerBound(1); inner <=
integers2d.GetUpperBound(1);
             inner++)
           Console.WriteLine($"      {'\u007b'}{outer}, {inner}{'\u007d'} =
" +
                             $"{integers2d.GetValue(outer, inner)}");
   }
}
// The example displays the following output:
//       Elements from index 0 to 9:
//          2, 4, 6, 8, 10, 12, 14, 16, 18, 20
//
//       Number of dimensions: 2
//          Dimension 0: from 0 to 7
//          Dimension 1: from 0 to 1
//          Values of array elements:
//             {0, 0} = 2
//             {0, 1} = 4
//             {1, 0} = 3
```

```
//              {1, 1} = 9
//              {2, 0} = 4
//              {2, 1} = 16
//              {3, 0} = 5
//              {3, 1} = 25
//              {4, 0} = 6
//              {4, 1} = 36
//              {5, 0} = 7
//              {5, 1} = 49
//              {6, 0} = 8
//              {6, 1} = 64
//              {7, 0} = 9
//              {7, 1} = 81
```

Remarks

`GetLowerBound(0)` returns the starting index of the first dimension of the array, and `GetLowerBound(Rank - 1)` returns the starting index of the last dimension of the array.

The GetLowerBound method always returns a value that indicates the index of the lower bound of the array, even if the array is empty.

Note that, although most arrays in .NET are zero-based (that is, the GetLowerBound method returns zero for each dimension of an array), .NET does support arrays that are not zero-based. Such arrays can be created with the CreateInstance(Type, Int32[], Int32[]) method, and can also be returned from unmanaged code.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetUpperBound(Int32)
- Rank

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.GetUpperBound(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the index of the last element of the specified dimension in the array.

```
C#
public int GetUpperBound (int dimension);
```

Parameters

`dimension` Int32

A zero-based dimension of the array whose upper bound needs to be determined.

Returns

Int32

The index of the last element of the specified dimension in the array, or -1 if the specified dimension is empty.

Exceptions

IndexOutOfRangeException

`dimension` is less than zero.

-or-

`dimension` is equal to or greater than Rank.

Examples

The following example uses the GetLowerBound and GetUpperBound methods to display the bounds of a one-dimensional and two-dimensional array and to display the values of their array elements.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Create a one-dimensional integer array.
      int[] integers = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
      // Get the upper and lower bound of the array.
      int upper = integers.GetUpperBound(0);
      int lower = integers.GetLowerBound(0);
      Console.WriteLine($"Elements from index {lower} to {upper}:");
      // Iterate the array.
      for (int ctr = lower; ctr <= upper; ctr++)
        Console.Write($"{(ctr == lower ?"    " : "")}{integers[ctr]}" +
                      $"{(ctr < upper ? ", " : Environment.NewLine)}");

      Console.WriteLine();

      // Create a two-dimensional integer array.
      int[,] integers2d= { {2, 4}, {3, 9}, {4, 16}, {5, 25},
                           {6, 36}, {7, 49}, {8, 64}, {9, 81} };
      // Get the number of dimensions.
      int rank = integers2d.Rank;
      Console.WriteLine($"Number of dimensions: {rank}");
      for (int ctr = 0; ctr < rank; ctr++)
        Console.WriteLine($"   Dimension {ctr}: " +
                          $"from {integers2d.GetLowerBound(ctr)} to
{integers2d.GetUpperBound(ctr)}");

      // Iterate the 2-dimensional array and display its values.
      Console.WriteLine("   Values of array elements:");
      for (int outer = integers2d.GetLowerBound(0); outer <=
integers2d.GetUpperBound(0);
           outer++)
        for (int inner = integers2d.GetLowerBound(1); inner <=
integers2d.GetUpperBound(1);
             inner++)
           Console.WriteLine($"      {'\u007b'}{outer}, {inner}{'\u007d'} =
" +
                             $"{integers2d.GetValue(outer, inner)}");
   }
}
// The example displays the following output:
//       Elements from index 0 to 9:
//          2, 4, 6, 8, 10, 12, 14, 16, 18, 20
//
//       Number of dimensions: 2
//          Dimension 0: from 0 to 7
//          Dimension 1: from 0 to 1
//          Values of array elements:
//             {0, 0} = 2
```

```
//              {0, 1} = 4
//              {1, 0} = 3
//              {1, 1} = 9
//              {2, 0} = 4
//              {2, 1} = 16
//              {3, 0} = 5
//              {3, 1} = 25
//              {4, 0} = 6
//              {4, 1} = 36
//              {5, 0} = 7
//              {5, 1} = 49
//              {6, 0} = 8
//              {6, 1} = 64
//              {7, 0} = 9
//              {7, 1} = 81
```

Remarks

`GetUpperBound(0)` returns the last index in the first dimension of the array, and `GetUpperBound(Rank - 1)` returns the last index of the last dimension of the array.

This method is an O(1) operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetLowerBound(Int32)
- Rank

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.GetValue Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified element in the current Array.

Overloads

Expand table

GetValue(Int32)	Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
GetValue(Int32[])	Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.
GetValue(Int64)	Gets the value at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.
GetValue(Int64[])	Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.
GetValue(Int32, Int32)	Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.
GetValue(Int64, Int64)	Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.
GetValue(Int32, Int32, Int32)	Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.
GetValue(Int64, Int64, Int64)	Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.

Examples

The following code example demonstrates how to set and get a specific value in a one-dimensional or multidimensional array.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Creates and initializes a one-dimensional array.
      String[] myArr1 = new String[5];

      // Sets the element at index 3.
      myArr1.SetValue( "three", 3 );
      Console.WriteLine( "[3]:   {0}", myArr1.GetValue( 3 ) );

      // Creates and initializes a two-dimensional array.
      String[,] myArr2 = new String[5,5];

      // Sets the element at index 1,3.
      myArr2.SetValue( "one-three", 1, 3 );
      Console.WriteLine( "[1,3]:   {0}", myArr2.GetValue( 1, 3 ) );

      // Creates and initializes a three-dimensional array.
      String[,,] myArr3 = new String[5,5,5];

      // Sets the element at index 1,2,3.
      myArr3.SetValue( "one-two-three", 1, 2, 3 );
      Console.WriteLine( "[1,2,3]:   {0}", myArr3.GetValue( 1, 2, 3 ) );

      // Creates and initializes a seven-dimensional array.
      String[,,,,,,] myArr7 = new String[5,5,5,5,5,5,5];

      // Sets the element at index 1,2,3,0,1,2,3.
      int[] myIndices = new int[7] { 1, 2, 3, 0, 1, 2, 3 };
      myArr7.SetValue( "one-two-three-zero-one-two-three", myIndices );
      Console.WriteLine( "[1,2,3,0,1,2,3]:   {0}", myArr7.GetValue(
myIndices ) );
   }
}


/*
This code produces the following output.

[3]:   three
[1,3]:   one-three
[1,2,3]:   one-two-three
[1,2,3,0,1,2,3]:   one-two-three-zero-one-two-three

*/
```

GetValue(Int32)

Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.

```
C#

public object? GetValue (int index);
```

Parameters

`index` Int32

A 32-bit integer that represents the position of the Array element to get.

Returns

Object

The value at the specified position in the one-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly one dimension.

IndexOutOfRangeException

`index` is outside the range of valid indexes for the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether the value of `index` is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetValue(Int32[])

Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.

```
C#

public object? GetValue (params int[] indices);
```

Parameters

`indices` Int32[]

A one-dimensional array of 32-bit integers that represent the indexes specifying the position of the Array element to get.

Returns

Object

The value at the specified position in the multidimensional Array.

Exceptions

ArgumentNullException

`indices` is `null`.

ArgumentException

The number of dimensions in the current Array is not equal to the number of elements in `indices`.

IndexOutOfRangeException

Any element in `indices` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The number of elements in `indices` must equal the number of dimensions in the Array. All elements in the `indices` array must collectively specify the position of the desired element in the multidimensional Array.

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetValue(Int64)

Gets the value at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.

```
C#
```

```
public object? GetValue (long index);
```

Parameters

`index` Int64

A 64-bit integer that represents the position of the Array element to get.

Returns

Object

The value at the specified position in the one-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly one dimension.

ArgumentOutOfRangeException

`index` is outside the range of valid indexes for the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether the value of `index` is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetValue(Int64[])

Gets the value at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.

C#

```
public object? GetValue (params long[] indices);
```

Parameters

`indices` Int64[]

A one-dimensional array of 64-bit integers that represent the indexes specifying the position of the Array element to get.

Returns

Object

The value at the specified position in the multidimensional Array.

Exceptions

ArgumentNullException

`indices` is `null`.

ArgumentException

The number of dimensions in the current Array is not equal to the number of elements in `indices`.

ArgumentOutOfRangeException

Any element in `indices` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The number of elements in `indices` must equal the number of dimensions in the Array. All elements in the `indices` array must collectively specify the position of the desired element in the multidimensional Array.

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetValue(Int32, Int32)

Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.

C#

```
public object? GetValue (int index1, int index2);
```

Parameters

`index1` Int32

A 32-bit integer that represents the first-dimension index of the Array element to get.

`index2` Int32

A 32-bit integer that represents the second-dimension index of the Array element to get.

Returns

Object

The value at the specified position in the two-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly two dimensions.

IndexOutOfRangeException

Either `index1` or `index2` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetValue(Int64, Int64)

Gets the value at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.

C#

```
public object? GetValue (long index1, long index2);
```

Parameters

`index1` Int64

A 64-bit integer that represents the first-dimension index of the Array element to get.

`index2` Int64

A 64-bit integer that represents the second-dimension index of the Array element to get.

Returns

Object

The value at the specified position in the two-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly two dimensions.

ArgumentOutOfRangeException

Either `index1` or `index2` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetValue(Int32, Int32, Int32)

Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.

C#

```
public object? GetValue (int index1, int index2, int index3);
```

Parameters

`index1` Int32

A 32-bit integer that represents the first-dimension index of the Array element to get.

`index2` Int32

A 32-bit integer that represents the second-dimension index of the Array element to get.

`index3` Int32

A 32-bit integer that represents the third-dimension index of the Array element to get.

Returns

Object

The value at the specified position in the three-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly three dimensions.

IndexOutOfRangeException

`index1` or `index2` or `index3` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetValue(Int64, Int64, Int64)

Gets the value at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.

C#

```
public object? GetValue (long index1, long index2, long index3);
```

Parameters

index1 Int64

A 64-bit integer that represents the first-dimension index of the Array element to get.

index2 Int64

A 64-bit integer that represents the second-dimension index of the Array element to get.

index3 Int64

A 64-bit integer that represents the third-dimension index of the Array element to get.

Returns

Object

The value at the specified position in the three-dimensional Array.

Exceptions

ArgumentException

The current Array does not have exactly three dimensions.

ArgumentOutOfRangeException

`index1` or `index2` or `index3` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

This method is an O(1) operation.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- SetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for the specified object and returns the index of its first occurrence in a one-dimensional array or in a range of elements in the array.

Overloads

Expand table

IndexOf(Array, Object)	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.
IndexOf(Array, Object, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified index to the end of the array.
IndexOf(Array, Object, Int32, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of ifs first occurrence. The range extends from a specified index for a specified number of elements.
IndexOf<T>(T[], T, Int32)	Searches for the specified object in a range of elements of a one dimensional array, and returns the index of its first occurrence. The range extends from a specified index to the end of the array.
IndexOf<T>(T[], T, Int32, Int32)	Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified index for a specified number of elements.
IndexOf<T>(T[], T)	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.

IndexOf(Array, Object)

Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.

```
C#

public static int IndexOf (Array array, object? value);
```

Parameters

`array` Array

The one-dimensional array to search.

`value` Object

The object to locate in `array`.

Returns

Int32

The index of the first occurrence of `value` in `array`, if found; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

Examples

The example calls the following three overloads of the IndexOf method to find the index of a string in a string array:

- IndexOf(Array, Object), to determine the first occurrence of the string "the" in a string array.
- IndexOf(Array, Object, Int32), to determine the first occurrence of the string "the" in the fourth to the last elements of a string array.
- IndexOf(Array, Object, Int32, Int32), to determine the first occurrence of the string "the" in a string array from the element that follows the last successful match to the end of the array.

C#

```
// Create a string array with 3 elements having the same value.
String[] strings = { "the", "quick", "brown", "fox", "jumps",
                     "over", "the", "lazy", "dog", "in", "the",
                     "barn" };

// Display the elements of the array.
Console.WriteLine("The array contains the following values:");
for (int i = strings.GetLowerBound(0); i <= strings.GetUpperBound(0);
i++)
   Console.WriteLine("   [{0,2}]: {1}", i, strings[i]);

// Search for the first occurrence of the duplicated value.
string searchString = "the";
int index = Array.IndexOf(strings, searchString);
Console.WriteLine("The first occurrence of \"{0}\" is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in the last
section of the array.
index = Array.IndexOf(strings, searchString, 4);
Console.WriteLine("The first occurrence of \"{0}\" between index 4 and
the end is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in a section
of the array.
int position = index + 1;
index = Array.IndexOf(strings, searchString, position,
strings.GetUpperBound(0) - position + 1);
Console.WriteLine("The first occurrence of \"{0}\" between index {1} and
index {2} is at index {3}.",
                  searchString, position, strings.GetUpperBound(0),
index);

// The example displays the following output:
//    The array contains the following values:
//       [ 0]: the
//       [ 1]: quick
//       [ 2]: brown
//       [ 3]: fox
//       [ 4]: jumps
//       [ 5]: over
//       [ 6]: the
//       [ 7]: lazy
//       [ 8]: dog
//       [ 9]: in
//       [10]: the
//       [11]: barn
//    The first occurrence of "the" is at index 0.
//    The first occurrence of "the" between index 4 and the end is at
index 6.
```

```
//    The first occurrence of "the" between index 7 and index 11 is at
index 10.
```

Remarks

This method searches all the elements of a one-dimensional array for `value`. To determine whether `value` exists in `array`, the method performs an equality comparison by calling each element's `Equals` method until it finds a match. This means that if the element overrides the Object.Equals(Object) method, that override is called.

Because most arrays have a lower bound of zero, this method generally returns -1 if `value` isn't found. In the rare case that the lower bound of the array is equal to Int32.MinValue(0x80000000) and `value` isn't found, this method returns Int32.MaxValue (0x7FFFFFFF).

This method is an O(`n`) operation, where `n` is the Length of `array`.

See also

- LastIndexOf

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(Array, Object, Int32)

Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified

index to the end of the array.

```
C#
public static int IndexOf (Array array, object? value, int startIndex);
```

Parameters

`array` Array

The one-dimensional array to search.

`value` Object

The object to locate in `array`.

`startIndex` Int32

The starting index of the search. 0 (zero) is valid in an empty array.

Returns

Int32

The index of the first occurrence of `value`, if it's found, within the range of elements in `array` that extends from `startIndex` to the last element; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

RankException

`array` is multidimensional.

Examples

The example calls the following three overloads of the IndexOf method to find the index of a string in a string array:

- IndexOf(Array, Object), to determine the first occurrence of the string "the" in a string array.
- IndexOf(Array, Object, Int32), to determine the first occurrence of the string "the" in the fourth to the last elements of a string array.
- IndexOf(Array, Object, Int32, Int32), to determine the first occurrence of the string "the" in a string array from the element that follows the last successful match to the end of the array.

C#

```
// Create a string array with 3 elements having the same value.
String[] strings = { "the", "quick", "brown", "fox", "jumps",
                     "over", "the", "lazy", "dog", "in", "the",
                     "barn" };

// Display the elements of the array.
Console.WriteLine("The array contains the following values:");
for (int i = strings.GetLowerBound(0); i <= strings.GetUpperBound(0);
i++)
   Console.WriteLine("   [{0,2}]: {1}", i, strings[i]);

// Search for the first occurrence of the duplicated value.
string searchString = "the";
int index = Array.IndexOf(strings, searchString);
Console.WriteLine("The first occurrence of \"{0}\" is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in the last
section of the array.
index = Array.IndexOf(strings, searchString, 4);
Console.WriteLine("The first occurrence of \"{0}\" between index 4 and
the end is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in a section
of the array.
int position = index + 1;
index = Array.IndexOf(strings, searchString, position,
strings.GetUpperBound(0) - position + 1);
Console.WriteLine("The first occurrence of \"{0}\" between index {1} and
index {2} is at index {3}.",
                  searchString, position, strings.GetUpperBound(0),
index);

// The example displays the following output:
//    The array contains the following values:
//       [ 0]: the
//       [ 1]: quick
//       [ 2]: brown
//       [ 3]: fox
```

```
//       [ 4]: jumps
//       [ 5]: over
//       [ 6]: the
//       [ 7]: lazy
//       [ 8]: dog
//       [ 9]: in
//       [10]: the
//       [11]: barn
//    The first occurrence of "the" is at index 0.
//    The first occurrence of "the" between index 4 and the end is at
index 6.
//    The first occurrence of "the" between index 7 and index 11 is at
index 10.
```

Remarks

This method searches a one-dimensional array from the element at index `startIndex` to the last element. To determine whether `value` exists in `array`, the method performs an equality comparison by calling the `Equals` method of every element until it finds a match. This means that if the element overrides the Object.Equals(Object) method, that override is called.

Because most arrays have a lower bound of zero, this method generally returns -1 if `value` isn't found. In the rare case that the lower bound of the array is equal to Int32.MinValue(0x80000000) and `value` isn't found, this method returns Int32.MaxValue (0x7FFFFFFF).

If `startIndex` equals Array.Length,the method returns -1. If `startIndex` is greater than Array.Length, the method throws an ArgumentOutOfRangeException.

This method is an O(`n`) operation, where `n` is the number of elements from `startIndex` to the end of `array`.

See also

- LastIndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(Array, Object, Int32, Int32)

Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of ifs first occurrence. The range extends from a specified index for a specified number of elements.

C#

```
public static int IndexOf (Array array, object? value, int startIndex,
int count);
```

Parameters

`array` Array

The one-dimensional array to search.

`value` Object

The object to locate in `array`.

`startIndex` Int32

The starting index of the search. 0 (zero) is valid in an empty array.

`count` Int32

The number of elements to search.

Returns

Int32

The index of the first occurrence of `value`, if it's found in the `array` from index `startIndex` to `startIndex` + `count` - 1; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

RankException

`array` is multidimensional.

Examples

The example calls the following three overloads of the IndexOf method to find the index of a string in a string array:

- IndexOf(Array, Object), to determine the first occurrence of the string "the" in a string array.
- IndexOf(Array, Object, Int32), to determine the first occurrence of the string "the" in the fourth to the last elements of a string array.
- IndexOf(Array, Object, Int32, Int32), to determine the first occurrence of the string "the" in a string array from the element that follows the last successful match to the end of the array. To determine the value of the `count` argument, it subtracts the upper bound of the array from the starting index and adds one.

C#

```
// Create a string array with 3 elements having the same value.
String[] strings = { "the", "quick", "brown", "fox", "jumps",
                     "over", "the", "lazy", "dog", "in", "the",
```

```
                      "barn" };

// Display the elements of the array.
Console.WriteLine("The array contains the following values:");
for (int i = strings.GetLowerBound(0); i <= strings.GetUpperBound(0);
i++)
   Console.WriteLine("   [{0,2}]: {1}", i, strings[i]);

// Search for the first occurrence of the duplicated value.
string searchString = "the";
int index = Array.IndexOf(strings, searchString);
Console.WriteLine("The first occurrence of \"{0}\" is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in the last
section of the array.
index = Array.IndexOf(strings, searchString, 4);
Console.WriteLine("The first occurrence of \"{0}\" between index 4 and
the end is at index {1}.",
                  searchString, index);

// Search for the first occurrence of the duplicated value in a section
of the array.
int position = index + 1;
index = Array.IndexOf(strings, searchString, position,
strings.GetUpperBound(0) - position + 1);
Console.WriteLine("The first occurrence of \"{0}\" between index {1} and
index {2} is at index {3}.",
                  searchString, position, strings.GetUpperBound(0),
index);

// The example displays the following output:
//    The array contains the following values:
//       [ 0]: the
//       [ 1]: quick
//       [ 2]: brown
//       [ 3]: fox
//       [ 4]: jumps
//       [ 5]: over
//       [ 6]: the
//       [ 7]: lazy
//       [ 8]: dog
//       [ 9]: in
//       [10]: the
//       [11]: barn
//    The first occurrence of "the" is at index 0.
//    The first occurrence of "the" between index 4 and the end is at
index 6.
//    The first occurrence of "the" between index 7 and index 11 is at
index 10.
```

Remarks

This method searches the elements of a one-dimensional array from `startIndex` to `startIndex` plus `count` minus 1, if `count` is greater than 0. To determine whether `value` exists in `array`, the method performs an equality comparison by calling the `Equals` method of every element until it finds a match. This means that if the element overrides the Object.Equals method, that override is called.

Because most arrays have a lower bound of zero, this method generally returns -1 when `value` isn't found. In the rare case that the lower bound of the array is equal to Int32.MinValue (0x80000000) and `value` isn't found, this method returns Int32.MaxValue (0x7FFFFFFF).

If `startindex` equals Array.Length, the method returns -1. If `startIndex` is greater than Array.Length, the method throws an ArgumentOutOfRangeException.

This method is an O(`n`) operation, where `n` is `count`.

See also

- LastIndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf<T>(T[], T, Int32)

Searches for the specified object in a range of elements of a one dimensional array, and returns the index of its first occurrence. The range extends from a specified index to the end of the array.

```
C#
public static int IndexOf<T> (T[] array, T value, int startIndex);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based array to search.

`value` T

The object to locate in `array`.

`startIndex` Int32

The zero-based starting index of the search. 0 (zero) is valid in an empty array.

Returns

Int32

The zero-based index of the first occurrence of `value` within the range of elements in `array` that extends from `startIndex` to the last element, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

Examples

The following example demonstrates all three generic overloads of the IndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The IndexOf<T>(T[], T) method overload searches the array from the beginning, and finds the first occurrence of the string. The

IndexOf<T>(T[], T, Int32) method overload is used to search the array beginning with index location 3 and continuing to the end of the array, and finds the second occurrence of the string. Finally, the IndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of two entries, beginning at index location two; it returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 2, 2): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2));

/* This code example produces the following output:

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Brachiosaurus
Deinonychus
Tyrannosaurus
Compsognathus

Array.IndexOf(dinosaurs, "Tyrannosaurus"): 0

Array.IndexOf(dinosaurs, "Tyrannosaurus", 3): 5

Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2): -1
*/
```

Remarks

This method searches a one-dimensional array from the element at `startIndex` to the end of the array. To determine whether `value` exists in `array`, the method performs an equality comparison by calling the `T.Equals` method on every element. This means that if `T` overrides the Equals method, that override is called.

If `startIndex` equals Length,the method returns -1. If `startIndex` is greater than Array.Length, the method throws an ArgumentOutOfRangeException.

This method is an O(`n`) operation, where `n` is the number of elements from `startIndex` to the end of `array`.

See also

- LastIndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf<T>(T[], T, Int32, Int32)

Searches for the specified object in a range of elements of a one-dimensional array, and returns the index of its first occurrence. The range extends from a specified index for a specified number of elements.

```
C#
```

```
public static int IndexOf<T> (T[] array, T value, int startIndex, int
count);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based array to search.

value T

The object to locate in `array`.

startIndex Int32

The zero-based starting index of the search. 0 (zero) is valid in an empty array.

count Int32

The number of elements in the section to search.

Returns

Int32

The zero-based index of the first occurrence of `value` within the range of elements in `array` that starts at `startIndex` and contains the number of elements specified in `count`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

Examples

The following example demonstrates all three generic overloads of the IndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The IndexOf<T>(T[], T) method overload searches the array from the beginning, and finds the first occurrence of the string. The IndexOf<T>(T[], T, Int32) method overload is used to search the array beginning with index location 3 and continuing to the end of the array, and finds the second occurrence of the string. Finally, the IndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of two entries, beginning at index location two; it returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 2, 2): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2));

/* This code example produces the following output:

Tyrannosaurus
Amargasaurus
Mamenchisaurus
```

```
Brachiosaurus
Deinonychus
Tyrannosaurus
Compsognathus

Array.IndexOf(dinosaurs, "Tyrannosaurus"): 0

Array.IndexOf(dinosaurs, "Tyrannosaurus", 3): 5

Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2): -1
*/
```

Remarks

This method searches the elements of a one-dimensional array from `startIndex` to `startIndex` plus `count` minus 1, if `count` is greater than 0. To determine whether `value` exists in `array`, the method performs an equality comparison by calling the `T.Equals` method on every element. This means that if `T` overrides the Equals method, that override is called.

If `startIndex` equals Array.Length, the method returns -1. If `startIndex` is greater than Array.Length, the method throws an ArgumentOutOfRangeException.

This method is an O(`n`) operation, where `n` is `count`.

See also

- LastIndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf<T>(T[], T)

Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.

```
C#

public static int IndexOf<T> (T[] array, T value);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based array to search.

`value` T

The object to locate in `array`.

Returns

Int32

The zero-based index of the first occurrence of `value` in the entire `array`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

Examples

The following example demonstrates all three generic overloads of the IndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The IndexOf<T>(T[], T) method overload searches the array from the beginning, and finds the first occurrence of the string. The IndexOf<T>(T[], T, Int32) method overload is used to search the array beginning with

index location 3 and continuing to the end of the array, and finds the second occurrence of the string. Finally, the IndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of two entries, beginning at index location two; it returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.IndexOf(dinosaurs, \"Tyrannosaurus\", 2, 2): {0}",
    Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2));

/* This code example produces the following output:

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Brachiosaurus
Deinonychus
Tyrannosaurus
Compsognathus

Array.IndexOf(dinosaurs, "Tyrannosaurus"): 0

Array.IndexOf(dinosaurs, "Tyrannosaurus", 3): 5

Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2): -1
*/
```

Remarks

This method searches all the elements of a one-dimensional array for `value`. To determine whether `value` exists in `array`, the method performs an equality comparison by calling the `T.Equals` method on every element. This means that if `T` overrides the Equals method, that override is called.

This method is an O(`n`) operation, where `n` is the Length of `array`.

See also

- LastIndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Initialize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes every element of the value-type Array by calling the parameterless constructor of the value type.

C#

```
public void Initialize ();
```

Remarks

This method is designed to help compilers support value-type arrays; most users do not need this method. It must not be used on reference-type arrays.

If the Array is not a value-type Array or if the value type does not have a parameterless constructor, the Array is not modified.

The value-type Array can have any lower bound and any number of dimensions.

This method is an O(`n`) operation, where `n` is Length.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.LastIndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the index of the last occurrence of a value in a one-dimensional Array or in a portion of the Array.

Overloads

Expand table

LastIndexOf(Array, Object)	Searches for the specified object and returns the index of the last occurrence within the entire one-dimensional Array.
LastIndexOf(Array, Object, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that extends from the first element to the specified index.
LastIndexOf(Array, Object, Int32, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that contains the specified number of elements and ends at the specified index.
LastIndexOf<T>(T[], T)	Searches for the specified object and returns the index of the last occurrence within the entire Array.
LastIndexOf<T>(T[], T, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.
LastIndexOf<T>(T[], T, Int32, Int32)	Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.

LastIndexOf(Array, Object)

Searches for the specified object and returns the index of the last occurrence within the entire one-dimensional Array.

```C#
public static int LastIndexOf (Array array, object? value);
```

Parameters

`array` Array

The one-dimensional Array to search.

`value` Object

The object to locate in `array`.

Returns

Int32

The index of the last occurrence of `value` within the entire `array`, if found; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

Examples

The following code example shows how to determine the index of the last occurrence of a specified element in an array.

```C#
// Creates and initializes a new Array with three elements of the same value.
Array myArray=Array.CreateInstance( typeof(string), 12 );
myArray.SetValue( "the", 0 );
myArray.SetValue( "quick", 1 );
myArray.SetValue( "brown", 2 );
myArray.SetValue( "fox", 3 );
myArray.SetValue( "jumps", 4 );
myArray.SetValue( "over", 5 );
myArray.SetValue( "the", 6 );
```

```
myArray.SetValue( "lazy", 7 );
myArray.SetValue( "dog", 8 );
myArray.SetValue( "in", 9 );
myArray.SetValue( "the", 10 );
myArray.SetValue( "barn", 11 );

// Displays the values of the Array.
Console.WriteLine( "The Array contains the following values:" );
PrintIndexAndValues( myArray );

// Searches for the last occurrence of the duplicated value.
string myString = "the";
int myIndex = Array.LastIndexOf( myArray, myString );
Console.WriteLine( "The last occurrence of \"{0}\" is at index {1}.",
myString, myIndex );

// Searches for the last occurrence of the duplicated value in the first
section of the Array.
myIndex = Array.LastIndexOf( myArray, myString, 8 );
Console.WriteLine( "The last occurrence of \"{0}\" between the start and
index 8 is at index {1}.", myString, myIndex );

// Searches for the last occurrence of the duplicated value in a section
of the Array.
// Note that the start index is greater than the end index because the
search is done backward.
myIndex = Array.LastIndexOf( myArray, myString, 10, 6 );
Console.WriteLine( "The last occurrence of \"{0}\" between index 5 and
index 10 is at index {1}.", myString, myIndex );

void PrintIndexAndValues( Array anArray )  {
   for ( int i = anArray.GetLowerBound(0); i <= anArray.GetUpperBound(0);
i++ )
      Console.WriteLine( "\t[{0}]:\t{1}", i, anArray.GetValue( i ) );
}

/*
This code produces the following output.

The Array contains the following values:
   [0]:    the
   [1]:    quick
   [2]:    brown
   [3]:    fox
   [4]:    jumps
   [5]:    over
   [6]:    the
   [7]:    lazy
   [8]:    dog
   [9]:    in
   [10]:    the
   [11]:    barn
The last occurrence of "the" is at index 10.
The last occurrence of "the" between the start and index 8 is at index 6.
```

```
The last occurrence of "the" between index 5 and index 10 is at index 10.
*/
```

Remarks

The one-dimensional Array is searched backward starting at the last element and ending at the first element.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

Since most arrays will have a lower bound of zero, this method would generally return -1 when `value` is not found. In the rare case that the lower bound of the array is equal to Int32.MinValue and `value` is not found, this method returns Int32.MaxValue, which is `System.Int32.MinValue - 1`.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In .NET Framework 2.0 and later versions, this method uses the Equals and CompareTo methods of the Array to determine whether the Object specified by the `value` parameter exists. In earlier versions of .NET Framework, this determination was made by using the Equals and CompareTo methods of the `value` Object itself.

CompareTo methods of the `item` parameter on the objects in the collection.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(Array, Object, Int32)

Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that extends from the first element to the specified index.

C#

```
public static int LastIndexOf (Array array, object? value, int
startIndex);
```

Parameters

`array` Array

The one-dimensional Array to search.

`value` Object

The object to locate in `array`.

`startIndex` Int32

The starting index of the backward search.

Returns

Int32

The index of the last occurrence of `value` within the range of elements in `array` that extends from the first element to `startIndex`, if found; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

RankException

`array` is multidimensional.

Examples

The following code example shows how to determine the index of the last occurrence of a specified element in an array.

C#

```
// Creates and initializes a new Array with three elements of the same
value.
Array myArray=Array.CreateInstance( typeof(string), 12 );
myArray.SetValue( "the", 0 );
myArray.SetValue( "quick", 1 );
myArray.SetValue( "brown", 2 );
myArray.SetValue( "fox", 3 );
myArray.SetValue( "jumps", 4 );
myArray.SetValue( "over", 5 );
myArray.SetValue( "the", 6 );
myArray.SetValue( "lazy", 7 );
myArray.SetValue( "dog", 8 );
myArray.SetValue( "in", 9 );
myArray.SetValue( "the", 10 );
myArray.SetValue( "barn", 11 );

// Displays the values of the Array.
Console.WriteLine( "The Array contains the following values:" );
PrintIndexAndValues( myArray );

// Searches for the last occurrence of the duplicated value.
string myString = "the";
int myIndex = Array.LastIndexOf( myArray, myString );
Console.WriteLine( "The last occurrence of \"{0}\" is at index {1}.",
myString, myIndex );

// Searches for the last occurrence of the duplicated value in the first
section of the Array.
myIndex = Array.LastIndexOf( myArray, myString, 8 );
Console.WriteLine( "The last occurrence of \"{0}\" between the start and
index 8 is at index {1}.", myString, myIndex );

// Searches for the last occurrence of the duplicated value in a section
of the Array.
// Note that the start index is greater than the end index because the
search is done backward.
myIndex = Array.LastIndexOf( myArray, myString, 10, 6 );
```

```
Console.WriteLine( "The last occurrence of \"{0}\" between index 5 and
index 10 is at index {1}.", myString, myIndex );

void PrintIndexAndValues( Array anArray )  {
   for ( int i = anArray.GetLowerBound(0); i <= anArray.GetUpperBound(0);
i++ )
      Console.WriteLine( "\t[{0}]:\t{1}", i, anArray.GetValue( i ) );
}

/*
This code produces the following output.

The Array contains the following values:
   [0]:    the
   [1]:    quick
   [2]:    brown
   [3]:    fox
   [4]:    jumps
   [5]:    over
   [6]:    the
   [7]:    lazy
   [8]:    dog
   [9]:    in
   [10]:    the
   [11]:    barn
The last occurrence of "the" is at index 10.
The last occurrence of "the" between the start and index 8 is at index 6.
The last occurrence of "the" between index 5 and index 10 is at index 10.
*/
```

Remarks

The one-dimensional Array is searched backward starting at `startIndex` and ending at the first element.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

Since most arrays will have a lower bound of zero, this method would generally return -1 when `value` is not found. In the rare case that the lower bound of the array is equal to Int32.MinValue and `value` is not found, this method returns Int32.MaxValue, which is `System.Int32.MinValue - 1`.

This method is an O(`n`) operation, where `n` is the number of elements from the beginning of `array` to `startIndex`.

In .NET Framework 2.0 and later versions, this method uses the Equals and CompareTo methods of the Array to determine whether the Object specified by the `value` parameter exists. In earlier versions of .NET Framework, this determination was made by using the Equals and CompareTo methods of the `value` Object itself.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(Array, Object, Int32, Int32)

Searches for the specified object and returns the index of the last occurrence within the range of elements in the one-dimensional Array that contains the specified number of elements and ends at the specified index.

C#

```
public static int LastIndexOf (Array array, object? value, int
startIndex, int count);
```

Parameters

`array` Array

The one-dimensional Array to search.

`value` Object

The object to locate in `array`.

`startIndex` Int32

The starting index of the backward search.

`count` Int32

The number of elements in the section to search.

Returns

Int32

The index of the last occurrence of `value` within the range of elements in `array` that contains the number of elements specified in `count` and ends at `startIndex`, if found; otherwise, the lower bound of the array minus 1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

RankException

`array` is multidimensional.

Examples

The following code example shows how to determine the index of the last occurrence of a specified element in an array. Note that the LastIndexOf method is a backward search; therefore, `count` must be less than or equal to (`startIndex` minus the lower bound of the array plus 1).

C#

```
// Creates and initializes a new Array with three elements of the same
value.
Array myArray=Array.CreateInstance( typeof(string), 12 );
myArray.SetValue( "the", 0 );
myArray.SetValue( "quick", 1 );
myArray.SetValue( "brown", 2 );
myArray.SetValue( "fox", 3 );
myArray.SetValue( "jumps", 4 );
myArray.SetValue( "over", 5 );
myArray.SetValue( "the", 6 );
myArray.SetValue( "lazy", 7 );
myArray.SetValue( "dog", 8 );
myArray.SetValue( "in", 9 );
myArray.SetValue( "the", 10 );
myArray.SetValue( "barn", 11 );

// Displays the values of the Array.
Console.WriteLine( "The Array contains the following values:" );
PrintIndexAndValues( myArray );

// Searches for the last occurrence of the duplicated value.
string myString = "the";
int myIndex = Array.LastIndexOf( myArray, myString );
Console.WriteLine( "The last occurrence of \"{0}\" is at index {1}.",
myString, myIndex );

// Searches for the last occurrence of the duplicated value in the first
section of the Array.
myIndex = Array.LastIndexOf( myArray, myString, 8 );
Console.WriteLine( "The last occurrence of \"{0}\" between the start and
index 8 is at index {1}.", myString, myIndex );

// Searches for the last occurrence of the duplicated value in a section
of the Array.
// Note that the start index is greater than the end index because the
search is done backward.
myIndex = Array.LastIndexOf( myArray, myString, 10, 6 );
Console.WriteLine( "The last occurrence of \"{0}\" between index 5 and
index 10 is at index {1}.", myString, myIndex );

void PrintIndexAndValues( Array anArray )  {
   for ( int i = anArray.GetLowerBound(0); i <= anArray.GetUpperBound(0);
i++ )
      Console.WriteLine( "\t[{0}]:\t{1}", i, anArray.GetValue( i ) );
}

/*
This code produces the following output.

The Array contains the following values:
   [0]:    the
   [1]:    quick
```

```
   [2]:    brown
   [3]:    fox
   [4]:    jumps
   [5]:    over
   [6]:    the
   [7]:    lazy
   [8]:    dog
   [9]:    in
   [10]:    the
   [11]:    barn
The last occurrence of "the" is at index 10.
The last occurrence of "the" between the start and index 8 is at index 6.
The last occurrence of "the" between index 5 and index 10 is at index 10.
*/
```

Remarks

The one-dimensional Array is searched backward starting at `startIndex` and ending at `startIndex` minus `count` plus 1, if `count` is greater than 0.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

Since most arrays will have a lower bound of zero, this method would generally return -1 when `value` is not found. In the rare case that the lower bound of the array is equal to Int32.MinValue and `value` is not found, this method returns Int32.MaxValue, which is `System.Int32.MinValue - 1`.

This method is an O(`n`) operation, where `n` is `count`.

In .NET Framework 2.0 and later versions, this method uses the Equals and CompareTo methods of the Array to determine whether the Object specified by the `value` parameter exists. In earlier versions of .NET Framework, this determination was made by using the Equals and CompareTo methods of the `value` Object itself.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf<T>(T[], T)

Searches for the specified object and returns the index of the last occurrence within the entire Array.

```
C#

public static int LastIndexOf<T> (T[] array, T value);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`value` T

The object to locate in `array`.

Returns

Int32

The zero-based index of the last occurrence of `value` within the entire `array`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

Examples

The following code example demonstrates all three generic overloads of the LastIndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The LastIndexOf<T>(T[], T) method overload searches the entire array from the end, and finds the second occurrence of the string. The LastIndexOf<T>(T[], T, Int32) method overload is used to search the array backward beginning with index location 3 and continuing to the beginning of the array, and finds the first occurrence of the string. Finally, the LastIndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of four entries, beginning at index location 4 and extending backward (that is, it searches the items at locations 4, 3, 2, and 1); this search returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 4, 4): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4));

/* This code example produces the following output:

Tyrannosaurus
```

```
Amargasaurus
Mamenchisaurus
Brachiosaurus
Deinonychus
Tyrannosaurus
Compsognathus

Array.LastIndexOf(dinosaurs, "Tyrannosaurus"): 5

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3): 0

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4): -1
*/
```

Remarks

The Array is searched backward starting at the last element and ending at the first element.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

This method is an O(`n`) operation, where `n` is the Length of `array`.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf<T>(T[], T, Int32)

Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that extends from the first element to the specified index.

```
C#
public static int LastIndexOf<T> (T[] array, T value, int startIndex);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to search.

`value` T

The object to locate in `array`.

`startIndex` Int32

The zero-based starting index of the backward search.

Returns

Int32

The zero-based index of the last occurrence of `value` within the range of elements in `array` that extends from the first element to `startIndex`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

Examples

The following code example demonstrates all three generic overloads of the LastIndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The LastIndexOf<T>(T[], T) method overload searches the entire array from the end, and finds the second occurrence of the string. The LastIndexOf<T>(T[], T, Int32) method overload is used to search the array backward beginning with index location 3 and continuing to the beginning of the array, and finds the first occurrence of the string. Finally, the LastIndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of four entries, beginning at index location 4 and extending backward (that is, it searches the items at locations 4, 3, 2, and 1); this search returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 4, 4): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4));

/* This code example produces the following output:

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Brachiosaurus
Deinonychus
Tyrannosaurus
```

```
Compsognathus

Array.LastIndexOf(dinosaurs, "Tyrannosaurus"): 5

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3): 0

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4): -1
*/
```

Remarks

The Array is searched backward starting at `startIndex` and ending at the first element.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

This method is an O(`n`) operation, where `n` is the number of elements from the beginning of `array` to `startIndex`.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf<T>(T[], T, Int32, Int32)

Searches for the specified object and returns the index of the last occurrence within the range of elements in the Array that contains the specified number of elements and ends at the specified index.

```
C#

public static int LastIndexOf<T> (T[] array, T value, int startIndex, int
count);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based Array to search.

value T

The object to locate in `array`.

startIndex Int32

The zero-based starting index of the backward search.

count Int32

The number of elements in the section to search.

Returns

Int32

The zero-based index of the last occurrence of `value` within the range of elements in `array` that contains the number of elements specified in `count` and ends at `startIndex`, if found; otherwise, -1.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`startIndex` is outside the range of valid indexes for `array`.

-or-

`count` is less than zero.

-or-

`startIndex` and `count` do not specify a valid section in `array`.

Examples

The following code example demonstrates all three generic overloads of the LastIndexOf method. An array of strings is created, with one entry that appears twice, at index location 0 and index location 5. The LastIndexOf<T>(T[], T) method overload searches the entire array from the end, and finds the second occurrence of the string. The LastIndexOf<T>(T[], T, Int32) method overload is used to search the array backward beginning with index location 3 and continuing to the beginning of the array, and finds the first occurrence of the string. Finally, the LastIndexOf<T>(T[], T, Int32, Int32) method overload is used to search a range of four entries, beginning at index location 4 and extending backward (that is, it searches the items at locations 4, 3, 2, and 1); this search returns -1 because there are no instances of the search string in that range.

C#

```
string[] dinosaurs = { "Tyrannosaurus",
    "Amargasaurus",
    "Mamenchisaurus",
    "Brachiosaurus",
    "Deinonychus",
    "Tyrannosaurus",
    "Compsognathus" };

Console.WriteLine();
foreach(string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\"): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus"));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 3): {0}",
```

```
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3));

Console.WriteLine(
    "\nArray.LastIndexOf(dinosaurs, \"Tyrannosaurus\", 4, 4): {0}",
    Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4));

/* This code example produces the following output:

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Brachiosaurus
Deinonychus
Tyrannosaurus
Compsognathus

Array.LastIndexOf(dinosaurs, "Tyrannosaurus"): 5

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 3): 0

Array.LastIndexOf(dinosaurs, "Tyrannosaurus", 4, 4): -1
*/
```

Remarks

The Array is searched backward starting at `startIndex` and ending at `startIndex` minus `count` plus 1, if `count` is greater than 0.

The elements are compared to the specified value using the Object.Equals method. If the element type is a nonintrinsic (user-defined) type, the `Equals` implementation of that type is used.

This method is an O(`n`) operation, where `n` is `count`.

See also

- IndexOf
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Resize<T>(T[], Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Changes the number of elements of a one-dimensional array to the specified new size.

C#

```
public static void Resize<T> (ref T[]? array, int newSize);
```

Type Parameters

T

The type of the elements of the array.

Parameters

array T[]

The one-dimensional, zero-based array to resize, or `null` to create a new array with the specified size.

newSize Int32

The size of the new array.

Exceptions

ArgumentOutOfRangeException

`newSize` is less than zero.

Examples

The following example shows how resizing affects the array.

C#

```
using System;

public class SamplesArray
{
    public static void Main()  {

        // Create and initialize a new string array.
        String[] myArr = {"The", "quick", "brown", "fox", "jumps",
            "over", "the", "lazy", "dog"};

        // Display the values of the array.
        Console.WriteLine(
            "The string array initially contains the following values:");
        PrintIndexAndValues(myArr);

        // Resize the array to a bigger size (five elements larger).
        Array.Resize(ref myArr, myArr.Length + 5);

        // Display the values of the array.
        Console.WriteLine("After resizing to a larger size, ");
        Console.WriteLine("the string array contains the following
values:");
        PrintIndexAndValues(myArr);

        // Resize the array to a smaller size (four elements).
        Array.Resize(ref myArr, 4);

        // Display the values of the array.
        Console.WriteLine("After resizing to a smaller size, ");
        Console.WriteLine("the string array contains the following
values:");
        PrintIndexAndValues(myArr);
    }

    public static void PrintIndexAndValues(String[] myArr)  {
        for(int i = 0; i < myArr.Length; i++)
        {
            Console.WriteLine("   [{0}] : {1}", i, myArr[i]);
        }
        Console.WriteLine();
    }
}

/*
This code produces the following output.

The string array initially contains the following values:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
```

```
   [7] : lazy
   [8] : dog

After resizing to a larger size,
the string array contains the following values:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog
   [9] :
   [10] :
   [11] :
   [12] :
   [13] :

After resizing to a smaller size,
the string array contains the following values:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox

*/
```

Remarks

This method allocates a new array with the specified size, copies elements from the old array to the new one, and then replaces the old array with the new one. `array` must be a one-dimensional array.

If `array` is `null`, this method creates a new array with the specified size.

If `newSize` is greater than the Length of the old array, a new array is allocated and all the elements are copied from the old array to the new one. If `newSize` is less than the Length of the old array, a new array is allocated and elements are copied from the old array to the new one until the new one is filled; the rest of the elements in the old array are ignored. If `newSize` is equal to the Length of the old array, this method does nothing.

This method is an O(`n`) operation, where `n` is `newSize`.

The Resize method resizes a one-dimensional array only. The Array class does not include a method for resizing multi-dimensional arrays. To do this, you must either

provide your own code or call a special-purpose method in a third-party library. The following code illustrates one possible implementation for a method that resizes an array of *n* dimensions.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int[,] arr = new int[10,2];
      for (int n1 = 0; n1 <= arr.GetUpperBound(0); n1++) {
         arr[n1, 0] = n1;
         arr[n1, 1] = n1 * 2;
      }

      // Make a 2-D array larger in the first dimension.
      arr = (int[,]) ResizeArray(arr, new int[] { 12, 2} );
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
         Console.WriteLine("{0}: {1}, {2}", ctr, arr[ctr, 0], arr[ctr, 1]);
      Console.WriteLine();

      // Make a 2-D array smaller in the first dimension.
      arr = (int[,]) ResizeArray(arr, new int[] { 2, 2} );
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
         Console.WriteLine("{0}: {1}, {2}", ctr, arr[ctr, 0], arr[ctr, 1]);
   }

   private static Array ResizeArray(Array arr, int[] newSizes)
   {
      if (newSizes.Length != arr.Rank)
         throw new ArgumentException("arr must have the same number of
dimensions " +
                                     "as there are elements in newSizes",
"newSizes");

      var temp = Array.CreateInstance(arr.GetType().GetElementType(),
newSizes);
      int length = arr.Length <= temp.Length ? arr.Length : temp.Length;
      Array.ConstrainedCopy(arr, 0, temp, 0, length);
      return temp;
   }
}
// The example displays the following output:
//       0: 0, 0
//       1: 1, 2
//       2: 2, 4
//       3: 3, 6
//       4: 4, 8
//       5: 5, 10
//       6: 6, 12
```

```
//        7: 7, 14
//        8: 8, 16
//        9: 9, 18
//        10: 0, 0
//        11: 0, 0
//
//        0: 0, 0
//        1: 1, 2
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Reverse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reverses the order of the elements in a one-dimensional Array or in a portion of the Array.

Overloads

Expand table

Reverse(Array)	Reverses the sequence of the elements in the entire one-dimensional Array.
Reverse(Array, Int32, Int32)	Reverses the sequence of a subset of the elements in the one-dimensional Array.
Reverse<T>(T[])	Reverses the sequence of the elements in the one-dimensional generic array.
Reverse<T>(T[], Int32, Int32)	Reverses the sequence of a subset of the elements in the one-dimensional generic array.

Reverse(Array)

Reverses the sequence of the elements in the entire one-dimensional Array.

C#

```
public static void Reverse (Array array);
```

Parameters

`array` Array

The one-dimensional Array to reverse.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

Examples

The following code example shows how to reverse the sort of the values in an Array.

C#

```
using System;
public class SamplesArray  {

   public static void Main()  {

      // Creates and initializes a new Array.
      Array myArray=Array.CreateInstance( typeof(string), 9 );
      myArray.SetValue( "The", 0 );
      myArray.SetValue( "quick", 1 );
      myArray.SetValue( "brown", 2 );
      myArray.SetValue( "fox", 3 );
      myArray.SetValue( "jumps", 4 );
      myArray.SetValue( "over", 5 );
      myArray.SetValue( "the", 6 );
      myArray.SetValue( "lazy", 7 );
      myArray.SetValue( "dog", 8 );

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintIndexAndValues( myArray );

      // Reverses the sort of the values of the Array.
      Array.Reverse( myArray );

      // Displays the values of the Array.
      Console.WriteLine( "After reversing:" );
      PrintIndexAndValues( myArray );
   }

   public static void PrintIndexAndValues( Array myArray )  {
      for ( int i = myArray.GetLowerBound(0); i <=
myArray.GetUpperBound(0); i++ )
         Console.WriteLine( "\t[{0}]:\t{1}", i, myArray.GetValue( i ) );
   }
}
/*
```

```
This code produces the following output.

The Array initially contains the following values:
    [0]:    The
    [1]:    quick
    [2]:    brown
    [3]:    fox
    [4]:    jumps
    [5]:    over
    [6]:    the
    [7]:    lazy
    [8]:    dog
After reversing:
    [0]:    dog
    [1]:    lazy
    [2]:    the
    [3]:    over
    [4]:    jumps
    [5]:    fox
    [6]:    brown
    [7]:    quick
    [8]:    The
*/
```

Remarks

After a call to this method, the element at `myArray[i]`, where `i` is any index in the array, moves to `myArray[j]`, where `j` equals `(myArray.Length + myArray.GetLowerBound(0)) - (i - myArray.GetLowerBound(0)) - 1`.

This method is an O(`n`) operation, where `n` is the Length of `array`.

In F#, the Array.rev function can be used instead.

As the following example shows, the Reverse method can be used to reverse a jagged array. It initializes a jagged array with one element for each month of the current year in the current culture's calendar. Each element contains an array with as many elements as that month has days. The example displays the contents of the array, calls the Reverse method, and then displays the contents of the reversed array.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
      int nMonths =
DateTimeFormatInfo.CurrentInfo.Calendar.GetMonthsInYear(DateTime.Now.Year
);
      int[][] months = new int[nMonths][];

      // Populate elements with number of days in month.
      for (int ctr = 0; ctr <= months.GetUpperBound(0); ctr++) {
         int daysInMonth = DateTime.DaysInMonth(DateTime.Now.Year, ctr +
1);
         months[ctr] = new int[daysInMonth] ;
         for (int ctr2 = 1; ctr2 <= daysInMonth; ctr2++)
            months[ctr][ctr2 - 1] = ctr2;
      }

      foreach (var month in months) {
         foreach (var day in month)
            Console.Write("{0} ", day);

         Console.WriteLine();
      }
      Console.WriteLine();

      Console.WriteLine("About to reverse array.\n");
      Array.Reverse(months);

      foreach (var month in months) {
         foreach (var day in month)
            Console.Write("{0} ", day);

         Console.WriteLine();
      }
   }
}
// The example displays output similar to the following:
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//
//    About to reverse array.
//
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29
//    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Reverse(Array, Int32, Int32)

Reverses the sequence of a subset of the elements in the one-dimensional Array.

```
C#

public static void Reverse (Array array, int index, int length);
```

Parameters

array Array

The one-dimensional Array to reverse.

index Int32

The starting index of the section to reverse.

length Int32

The number of elements in the section to reverse.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

Examples

The following code example shows how to reverse the sort of the values in a range of elements in an Array.

```
C#
```

```
using System;
public class SamplesArray1  {

   public static void Main()  {

      // Creates and initializes a new Array.
      Array myArray=Array.CreateInstance( typeof(string), 9 );
      myArray.SetValue( "The", 0 );
      myArray.SetValue( "QUICK", 1 );
      myArray.SetValue( "BROWN", 2 );
      myArray.SetValue( "FOX", 3 );
      myArray.SetValue( "jumps", 4 );
      myArray.SetValue( "over", 5 );
      myArray.SetValue( "the", 6 );
      myArray.SetValue( "lazy", 7 );
      myArray.SetValue( "dog", 8 );

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintIndexAndValues( myArray );

      // Reverses the sort of the values of the Array.
      Array.Reverse( myArray, 1, 3 );

      // Displays the values of the Array.
      Console.WriteLine( "After reversing:" );
      PrintIndexAndValues( myArray );
   }

   public static void PrintIndexAndValues( Array myArray )  {
      for ( int i = myArray.GetLowerBound(0); i <=
myArray.GetUpperBound(0); i++ )
         Console.WriteLine( "\t[{0}]:\t{1}", i, myArray.GetValue( i ) );
   }
}
/*
This code produces the following output.

The Array initially contains the following values:
    [0]:    The
    [1]:    QUICK
    [2]:    BROWN
    [3]:    FOX
    [4]:    jumps
    [5]:    over
    [6]:    the
    [7]:    lazy
    [8]:    dog
After reversing:
    [0]:    The
    [1]:    FOX
    [2]:    BROWN
    [3]:    QUICK
```

```
    [4]:    jumps
    [5]:    over
    [6]:    the
    [7]:    lazy
    [8]:    dog
*/
```

Remarks

After a call to this method, the element at `myArray[i]`, where `i` is any index in the array, moves to `myArray[j]`, where `j` equals `(myArray.Length + myArray.GetLowerBound(0)) - (i - myArray.GetLowerBound(0)) - 1`.

The Reverse method can be used to reverse a jagged array.

This method is an O(`n`) operation, where `n` is `length`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Reverse<T>(T[])

Reverses the sequence of the elements in the one-dimensional generic array.

C#

```
public static void Reverse<T> (T[] array);
```

Type Parameters

`T`

The type of the elements in `array`.

Parameters

`array` T[]

The one-dimensional array of elements to reverse.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Reverse<T>(T[], Int32, Int32)

Reverses the sequence of a subset of the elements in the one-dimensional generic array.

C#

```
public static void Reverse<T> (T[] array, int index, int length);
```

Type Parameters

`T`

The type of the elements in `array`.

Parameters

`array` T[]

The one-dimensional array of elements to reverse.

`index` Int32

The starting index of the section to reverse.

`length` Int32

The number of elements in the section to reverse.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.SetValue Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the specified element in the current Array to the specified value.

Overloads

Expand table

SetValue(Object, Int32)	Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
SetValue(Object, Int32[])	Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.
SetValue(Object, Int64[])	Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.
SetValue(Object, Int32, Int32)	Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.
SetValue(Object, Int64, Int64)	Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.
SetValue(Object, Int32, Int32, Int32)	Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.
SetValue(Object, Int64, Int64, Int64)	Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.
SetValue(Object, Int64)	Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.

Examples

The following code example demonstrates how to set and get a specific value in a one-dimensional or multidimensional array.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Creates and initializes a one-dimensional array.
      String[] myArr1 = new String[5];

      // Sets the element at index 3.
      myArr1.SetValue( "three", 3 );
      Console.WriteLine( "[3]:   {0}", myArr1.GetValue( 3 ) );

      // Creates and initializes a two-dimensional array.
      String[,] myArr2 = new String[5,5];

      // Sets the element at index 1,3.
      myArr2.SetValue( "one-three", 1, 3 );
      Console.WriteLine( "[1,3]:   {0}", myArr2.GetValue( 1, 3 ) );

      // Creates and initializes a three-dimensional array.
      String[,,] myArr3 = new String[5,5,5];

      // Sets the element at index 1,2,3.
      myArr3.SetValue( "one-two-three", 1, 2, 3 );
      Console.WriteLine( "[1,2,3]:   {0}", myArr3.GetValue( 1, 2, 3 ) );

      // Creates and initializes a seven-dimensional array.
      String[,,,,,,] myArr7 = new String[5,5,5,5,5,5,5];

      // Sets the element at index 1,2,3,0,1,2,3.
      int[] myIndices = new int[7] { 1, 2, 3, 0, 1, 2, 3 };
      myArr7.SetValue( "one-two-three-zero-one-two-three", myIndices );
      Console.WriteLine( "[1,2,3,0,1,2,3]:   {0}", myArr7.GetValue(
myIndices ) );
   }
}


/*
This code produces the following output.

[3]:   three
[1,3]:   one-three
[1,2,3]:   one-two-three
[1,2,3,0,1,2,3]:   one-two-three-zero-one-two-three

*/
```

SetValue(Object, Int32)

Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.

```
C#

public void SetValue (object? value, int index);
```

Parameters

`value` Object

The new value for the specified element.

`index` Int32

A 32-bit integer that represents the position of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly one dimension.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

IndexOutOfRangeException

`index` is outside the range of valid indexes for the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether the value of `index` is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> ⓘ **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null

reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

SetValue(Object, Int32[])

Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 32-bit integers.

C#

```
public void SetValue (object? value, params int[] indices);
```

Parameters

`value` Object

The new value for the specified element.

`indices` Int32[]

A one-dimensional array of 32-bit integers that represent the indexes specifying the position of the element to set.

Exceptions

ArgumentNullException

`indices` is `null`.

ArgumentException

The number of dimensions in the current Array is not equal to the number of elements in `indices`.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

IndexOutOfRangeException

Any element in `indices` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The number of elements in `indices` must equal the number of dimensions in the Array. All elements in the `indices` array must collectively specify the position of the desired element in the multidimensional Array.

The GetLowerBound and GetUpperBound methods can determine whether any of the values in the `indices` array is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

SetValue(Object, Int64[])

Sets a value to the element at the specified position in the multidimensional Array. The indexes are specified as an array of 64-bit integers.

```
C#

public void SetValue (object? value, params long[] indices);
```

Parameters

`value` Object

The new value for the specified element.

`indices` Int64[]

A one-dimensional array of 64-bit integers that represent the indexes specifying the position of the element to set.

Exceptions

ArgumentNullException

`indices` is `null`.

ArgumentException

The number of dimensions in the current Array is not equal to the number of elements in `indices`.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

ArgumentOutOfRangeException

Any element in `indices` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The number of elements in `indices` must equal the number of dimensions in the Array. All elements in the `indices` array must collectively specify the position of the desired element in the multidimensional Array.

The GetLowerBound and GetUpperBound methods can determine whether any of the values in the `indices` array is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> ⓘ **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	2.0, 2.1

SetValue(Object, Int32, Int32)

Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 32-bit integers.

```
C#
public void SetValue (object? value, int index1, int index2);
```

Parameters

`value` Object

The new value for the specified element.

`index1` Int32

A 32-bit integer that represents the first-dimension index of the Array element to set.

`index2` Int32

A 32-bit integer that represents the second-dimension index of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly two dimensions.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

IndexOutOfRangeException

Either `index1` or `index2` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SetValue(Object, Int64, Int64)

Sets a value to the element at the specified position in the two-dimensional Array. The indexes are specified as 64-bit integers.

C#

```
public void SetValue (object? value, long index1, long index2);
```

Parameters

`value` Object

The new value for the specified element.

`index1` Int64

A 64-bit integer that represents the first-dimension index of the Array element to set.

`index2` Int64

A 64-bit integer that represents the second-dimension index of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly two dimensions.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

ArgumentOutOfRangeException

Either `index1` or `index2` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SetValue(Object, Int32, Int32, Int32)

Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 32-bit integers.

C#

```
public void SetValue (object? value, int index1, int index2, int index3);
```

Parameters

`value` Object

The new value for the specified element.

`index1` Int32

A 32-bit integer that represents the first-dimension index of the Array element to set.

`index2` Int32

A 32-bit integer that represents the second-dimension index of the Array element to set.

`index3` Int32

A 32-bit integer that represents the third-dimension index of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly three dimensions.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

IndexOutOfRangeException

`index1` or `index2` or `index3` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> ⓘ **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SetValue(Object, Int64, Int64, Int64)

Sets a value to the element at the specified position in the three-dimensional Array. The indexes are specified as 64-bit integers.

```
C#

public void SetValue (object? value, long index1, long index2, long
index3);
```

Parameters

`value` Object

The new value for the specified element.

`index1` Int64

A 64-bit integer that represents the first-dimension index of the Array element to set.

`index2` Int64

A 64-bit integer that represents the second-dimension index of the Array element to set.

`index3` Int64

A 64-bit integer that represents the third-dimension index of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly three dimensions.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

ArgumentOutOfRangeException

`index1` or `index2` or `index3` is outside the range of valid indexes for the corresponding dimension of the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether any of the indexes is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> ⓘ **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SetValue(Object, Int64)

Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 64-bit integer.

```
C#

public void SetValue (object? value, long index);
```

Parameters

`value` Object

The new value for the specified element.

`index` Int64

A 64-bit integer that represents the position of the Array element to set.

Exceptions

ArgumentException

The current Array does not have exactly one dimension.

InvalidCastException

`value` cannot be cast to the element type of the current Array.

ArgumentOutOfRangeException

`index` is outside the range of valid indexes for the current Array.

Remarks

The GetLowerBound and GetUpperBound methods can determine whether the value of `index` is out of bounds.

For more information about conversions, see Convert.

This method is an O(1) operation.

> ⓘ **Note**
>
> If **SetValue** is used to assign `null` to an element of an array of value types, all fields of the element are initialized to zero. The value of the element is not a null reference, and cannot be found by searching for a null reference.

See also

- GetLowerBound(Int32)
- GetUpperBound(Int32)
- GetValue

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.Sort Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sorts the elements in a one-dimensional array.

Overloads

Expand table

Sort(Array, Array, Int32, Int32, IComparer)	Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.
Sort(Array, Int32, Int32, IComparer)	Sorts the elements in a range of elements in a one-dimensional Array using the specified IComparer.
Sort(Array, Array, Int32, Int32)	Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.
Sort(Array, Int32, Int32)	Sorts the elements in a range of elements in a one-dimensional Array using the IComparable implementation of each element of the Array.
Sort(Array, Array, IComparer)	Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.
Sort(Array, Array)	Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.
Sort(Array)	Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
Sort(Array, IComparer)	Sorts the elements in a one-dimensional Array using the specified IComparer.

Sort<T>(T[])	Sorts the elements in an entire Array using the IComparable<T> generic interface implementation of each element of the Array.
Sort<T>(T[], IComparer<T>)	Sorts the elements in an Array using the specified IComparer<T> generic interface.
Sort<T>(T[], Comparison<T>)	Sorts the elements in an Array using the specified Comparison<T>.
Sort<T>(T[], Int32, Int32)	Sorts the elements in a range of elements in an Array using the IComparable<T> generic interface implementation of each element of the Array.
Sort<T>(T[], Int32, Int32, IComparer<T>)	Sorts the elements in a range of elements in an Array using the specified IComparer<T> generic interface.
Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>)	Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.
Sort<TKey,TValue>(TKey[], TValue[])	Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable<T> generic interface implementation of each key.
Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>)	Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.
Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32)	Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable<T> generic interface implementation of each key.

Sort(Array, Array, Int32, Int32, IComparer)

Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.

C#

```
public static void Sort (Array keys, Array? items, int index, int length,
System.Collections.IComparer? comparer);
```

Parameters

`keys` Array

The one-dimensional Array that contains the keys to sort.

`items` Array

The one-dimensional Array that contains the items that correspond to each of the keys in the `keys` Array.

-or-

`null` to sort only the `keys` Array.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

`comparer` IComparer

The IComparer implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Exceptions

ArgumentNullException

`keys` is `null`.

RankException

The `keys` Array is multidimensional.

-or-

The `items` Array is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `keys`.

-or-

`length` is less than zero.

ArgumentException

`items` is not `null`, and the lower bound of `keys` does not match the lower bound of `items`.

-or-

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

`index` and `length` do not specify a valid range in the `keys` Array.

-or-

`items` is not `null`, and `index` and `length` do not specify a valid range in the `items` Array.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException

`comparer` is `null`, and one or more elements in the `keys` Array do not implement the IComparable interface.

Examples

The following code example shows how to sort two associated arrays where the first array contains the keys and the second array contains the values. Sorts are done using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class SamplesArray  {

   public class myReverserClass : IComparer  {

      // Calls CaseInsensitiveComparer.Compare with the parameters
```

```
reversed.
      int IComparer.Compare( Object x, Object y )  {
          return( (new CaseInsensitiveComparer()).Compare( y, x ) );
      }
   }

   public static void Main()  {

      // Creates and initializes a new Array and a new custom comparer.
      String[] myKeys = { "red", "GREEN", "YELLOW", "BLUE", "purple",
"black", "orange" };
      String[] myValues = { "strawberries", "PEARS", "LIMES", "BERRIES",
"grapes", "olives", "cantaloupe" };
      IComparer myComparer = new myReverserClass();

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the default comparer.
      Array.Sort( myKeys, myValues, 1, 3 );
      Console.WriteLine( "After sorting a section of the Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, 1, 3, myComparer );
      Console.WriteLine( "After sorting a section of the Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the default comparer.
      Array.Sort( myKeys, myValues );
      Console.WriteLine( "After sorting the entire Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, myComparer );
      Console.WriteLine( "After sorting the entire Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );
   }

   public static void PrintKeysAndValues( String[] myKeys, String[]
myValues )  {
      for ( int i = 0; i < myKeys.Length; i++ )  {
         Console.WriteLine( "   {0,-10}: {1}", myKeys[i], myValues[i] );
      }
      Console.WriteLine();
   }
}
```

```
/*
This code produces the following output.

The Array initially contains the following values:
   red       : strawberries
   GREEN     : PEARS
   YELLOW    : LIMES
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the default comparer:
   red       : strawberries
   BLUE      : BERRIES
   GREEN     : PEARS
   YELLOW    : LIMES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the reverse case-insensitive
comparer:
   red       : strawberries
   YELLOW    : LIMES
   GREEN     : PEARS
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting the entire Array using the default comparer:
   black     : olives
   BLUE      : BERRIES
   GREEN     : PEARS
   orange    : cantaloupe
   purple    : grapes
   red       : strawberries
   YELLOW    : LIMES

After sorting the entire Array using the reverse case-insensitive
comparer:
   YELLOW    : LIMES
   red       : strawberries
   purple    : grapes
   orange    : cantaloupe
   GREEN     : PEARS
   BLUE      : BERRIES
   black     : olives

*/
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

If `comparer` is `null`, each key within the specified range of elements in the `keys` Array must implement the IComparable interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

.NET includes predefined IComparer implementations listed in the following table.

Expand table

Implementation	Description
System.Collections.CaseInsensitiveComparer	Compares any two objects, but performs a case-insensitive comparison of strings.
Comparer.Default	Compares any two objects by using the sorting conventions of the current culture.
Comparer.DefaultInvariant	Compares any two objects by using the sorting conventions of the invariant culture.
Comparer<T>.Default	Compares two objects of type `T` by using the type's default sort order.

You can also support custom comparisons by providing an instance of your own IComparer implementation to the `comparer` parameter. The example does this by defining a custom IComparer implementation that reverses the default sort order and performs case-insensitive string comparison.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.

- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer
- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, Int32, Int32, IComparer)

Sorts the elements in a range of elements in a one-dimensional Array using the specified IComparer.

```
C#

public static void Sort (Array array, int index, int length,
System.Collections.IComparer? comparer);
```

Parameters

`array` Array

The one-dimensional Array to sort.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

`comparer` IComparer

The IComparer implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException

`comparer` is `null`, and one or more elements in `array` do not implement the IComparable interface.

Examples

The following code example shows how to sort the values in an Array using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class ReverseComparer : IComparer
{
   // Call CaseInsensitiveComparer.Compare with the parameters reversed.
   public int Compare(Object x, Object y)
   {
       return (new CaseInsensitiveComparer()).Compare(y, x );
   }
}

public class Example
{
   public static void Main()
   {
      // Create and initialize a new array.
      String[] words = { "The", "QUICK", "BROWN", "FOX", "jumps",
                         "over", "the", "lazy", "dog" };
      // Instantiate the reverse comparer.
      IComparer revComparer = new ReverseComparer();

      // Display the values of the array.
      Console.WriteLine( "The original order of elements in the array:"
);
      DisplayValues(words);

      // Sort a section of the array using the default comparer.
      Array.Sort(words, 1, 3);
```

```
      Console.WriteLine( "After sorting elements 1-3 by using the default
comparer:");
      DisplayValues(words);

      // Sort a section of the array using the reverse case-insensitive
comparer.
      Array.Sort(words, 1, 3, revComparer);
      Console.WriteLine( "After sorting elements 1-3 by using the reverse
case-insensitive comparer:");
      DisplayValues(words);

      // Sort the entire array using the default comparer.
      Array.Sort(words);
      Console.WriteLine( "After sorting the entire array by using the
default comparer:");
      DisplayValues(words);

      // Sort the entire array by using the reverse case-insensitive
comparer.
      Array.Sort(words, revComparer);
      Console.WriteLine( "After sorting the entire array using the
reverse case-insensitive comparer:");
      DisplayValues(words);
   }

   public static void DisplayValues(String[] arr)
   {
      for ( int i = arr.GetLowerBound(0); i <= arr.GetUpperBound(0);
            i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, arr[i] );
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//    The original order of elements in the array:
//       [0] : The
//       [1] : QUICK
//       [2] : BROWN
//       [3] : FOX
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the default comparer:
//       [0] : The
//       [1] : BROWN
//       [2] : FOX
//       [3] : QUICK
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
```

```
//       [8] : dog
//
//    After sorting elements 1-3 by using the reverse case-insensitive
comparer:
//       [0] : The
//       [1] : QUICK
//       [2] : FOX
//       [3] : BROWN
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting the entire array by using the default comparer:
//       [0] : BROWN
//       [1] : dog
//       [2] : FOX
//       [3] : jumps
//       [4] : lazy
//       [5] : over
//       [6] : QUICK
//       [7] : the
//       [8] : The
//
//    After sorting the entire array using the reverse case-insensitive
comparer:
//       [0] : the
//       [1] : The
//       [2] : QUICK
//       [3] : over
//       [4] : lazy
//       [5] : jumps
//       [6] : FOX
//       [7] : dog
//       [8] : BROWN
```

Remarks

If `comparer` is `null`, each element within the specified range of elements in `array` must implement the IComparable interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

.NET includes predefined IComparer implementations listed in the following table.

Expand table

Implementation	Description
System.Collections.CaseInsensitiveComparer	Compares any two objects, but performs a case-insensitive comparison of strings.
Comparer.Default	Compares any two objects by using the sorting conventions of the current culture.
Comparer.DefaultInvariant	Compares any two objects by using the sorting conventions of the invariant culture.
Comparer<T>.Default	Compares two objects of type `T` by using the type's default sort order.

You can also support custom comparisons by providing an instance of your own IComparer implementation to the `comparer` parameter. The example does this by defining a `ReverseComparer` class that reverses the default sort order for instances of a type and performs case-insensitive string comparison.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer
- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, Array, Int32, Int32)

Sorts a range of elements in a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.

C#

```
public static void Sort (Array keys, Array? items, int index, int
length);
```

Parameters

keys Array

The one-dimensional Array that contains the keys to sort.

items Array

The one-dimensional Array that contains the items that correspond to each of the keys in the `keys` Array.

-or-

`null` to sort only the `keys` Array.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

Exceptions

ArgumentNullException

`keys` is `null`.

RankException

The `keys` Array is multidimensional.

-or-

The `items` Array is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `keys`.

-or-

`length` is less than zero.

ArgumentException

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

`index` and `length` do not specify a valid range in the `keys` Array.

-or-

`items` is not `null`, and `index` and `length` do not specify a valid range in the `items` Array.

InvalidOperationException

One or more elements in the `keys` Array do not implement the IComparable interface.

Examples

The following code example shows how to sort two associated arrays where the first array contains the keys and the second array contains the values. Sorts are done using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class SamplesArray  {

   public class myReverserClass : IComparer  {

      // Calls CaseInsensitiveComparer.Compare with the parameters
reversed.
      int IComparer.Compare( Object x, Object y )  {
          return( (new CaseInsensitiveComparer()).Compare( y, x ) );
      }
   }

   public static void Main()  {

      // Creates and initializes a new Array and a new custom comparer.
      String[] myKeys = { "red", "GREEN", "YELLOW", "BLUE", "purple",
"black", "orange" };
      String[] myValues = { "strawberries", "PEARS", "LIMES", "BERRIES",
"grapes", "olives", "cantaloupe" };
      IComparer myComparer = new myReverserClass();

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the default comparer.
      Array.Sort( myKeys, myValues, 1, 3 );
      Console.WriteLine( "After sorting a section of the Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, 1, 3, myComparer );
      Console.WriteLine( "After sorting a section of the Array using the
reverse case-insensitive comparer:" );
```

```
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the default comparer.
      Array.Sort( myKeys, myValues );
      Console.WriteLine( "After sorting the entire Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, myComparer );
      Console.WriteLine( "After sorting the entire Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );
   }

   public static void PrintKeysAndValues( String[] myKeys, String[]
myValues )  {
      for ( int i = 0; i < myKeys.Length; i++ )  {
         Console.WriteLine( "   {0,-10}: {1}", myKeys[i], myValues[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The Array initially contains the following values:
   red       : strawberries
   GREEN     : PEARS
   YELLOW    : LIMES
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the default comparer:
   red       : strawberries
   BLUE      : BERRIES
   GREEN     : PEARS
   YELLOW    : LIMES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the reverse case-insensitive
comparer:
   red       : strawberries
   YELLOW    : LIMES
   GREEN     : PEARS
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
```

```
   orange    : cantaloupe

After sorting the entire Array using the default comparer:
   black     : olives
   BLUE      : BERRIES
   GREEN     : PEARS
   orange    : cantaloupe
   purple    : grapes
   red       : strawberries
   YELLOW    : LIMES

After sorting the entire Array using the reverse case-insensitive
comparer:
   YELLOW    : LIMES
   red       : strawberries
   purple    : grapes
   orange    : cantaloupe
   GREEN     : PEARS
   BLUE      : BERRIES
   black     : olives

*/
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

Each key within the specified range of elements in the `keys` Array must implement the IComparable interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^{N}, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

See also

- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, Int32, Int32)

Sorts the elements in a range of elements in a one-dimensional Array using the IComparable implementation of each element of the Array.

C#

```
public static void Sort (Array array, int index, int length);
```

Parameters

`array` Array

The one-dimensional Array to sort.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

InvalidOperationException

One or more elements in `array` do not implement the IComparable interface.

Examples

The following code example shows how to sort the values in an Array using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

```C#
using System;
using System.Collections;

public class ReverseComparer : IComparer
{
   // Call CaseInsensitiveComparer.Compare with the parameters reversed.
   public int Compare(Object x, Object y)
   {
```

```
        return (new CaseInsensitiveComparer()).Compare(y, x );
    }
}

public class Example
{
   public static void Main()
   {
      // Create and initialize a new array.
      String[] words = { "The", "QUICK", "BROWN", "FOX", "jumps",
                         "over", "the", "lazy", "dog" };
      // Instantiate the reverse comparer.
      IComparer revComparer = new ReverseComparer();

      // Display the values of the array.
      Console.WriteLine( "The original order of elements in the array:"
);
      DisplayValues(words);

      // Sort a section of the array using the default comparer.
      Array.Sort(words, 1, 3);
      Console.WriteLine( "After sorting elements 1-3 by using the default
comparer:");
      DisplayValues(words);

      // Sort a section of the array using the reverse case-insensitive
comparer.
      Array.Sort(words, 1, 3, revComparer);
      Console.WriteLine( "After sorting elements 1-3 by using the reverse
case-insensitive comparer:");
      DisplayValues(words);

      // Sort the entire array using the default comparer.
      Array.Sort(words);
      Console.WriteLine( "After sorting the entire array by using the
default comparer:");
      DisplayValues(words);

      // Sort the entire array by using the reverse case-insensitive
comparer.
      Array.Sort(words, revComparer);
      Console.WriteLine( "After sorting the entire array using the
reverse case-insensitive comparer:");
      DisplayValues(words);
   }

   public static void DisplayValues(String[] arr)
   {
      for ( int i = arr.GetLowerBound(0); i <= arr.GetUpperBound(0);
            i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, arr[i] );
      }
      Console.WriteLine();
   }
}
```

```
// The example displays the following output:
//    The original order of elements in the array:
//       [0] : The
//       [1] : QUICK
//       [2] : BROWN
//       [3] : FOX
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the default comparer:
//       [0] : The
//       [1] : BROWN
//       [2] : FOX
//       [3] : QUICK
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the reverse case-insensitive
comparer:
//       [0] : The
//       [1] : QUICK
//       [2] : FOX
//       [3] : BROWN
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting the entire array by using the default comparer:
//       [0] : BROWN
//       [1] : dog
//       [2] : FOX
//       [3] : jumps
//       [4] : lazy
//       [5] : over
//       [6] : QUICK
//       [7] : the
//       [8] : The
//
//    After sorting the entire array using the reverse case-insensitive
comparer:
//       [0] : the
//       [1] : The
//       [2] : QUICK
//       [3] : over
//       [4] : lazy
//       [5] : jumps
//       [6] : FOX
```

```
//        [7] : dog
//        [8] : BROWN
```

Remarks

Each element within the specified range of elements in `array` must implement the IComparable interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

See also

- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, Array, IComparer)

Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer.

```
C#

public static void Sort (Array keys, Array? items,
System.Collections.IComparer? comparer);
```

Parameters

`keys` Array

The one-dimensional Array that contains the keys to sort.

`items` Array

The one-dimensional Array that contains the items that correspond to each of the keys in the `keys` Array.

-or-

`null` to sort only the `keys` Array.

`comparer` IComparer

The IComparer implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Exceptions

ArgumentNullException

`keys` is `null`.

RankException

The `keys` Array is multidimensional.

-or-

The `items` Array is multidimensional.

ArgumentException

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException

`comparer` is `null`, and one or more elements in the `keys` Array do not implement the IComparable interface.

Examples

The following example shows how to sort two associated arrays where the first array contains the keys and the second array contains the values. Sorts are done using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class SamplesArray  {

   public class myReverserClass : IComparer  {

      // Calls CaseInsensitiveComparer.Compare with the parameters
reversed.
      int IComparer.Compare( Object x, Object y )  {
          return( (new CaseInsensitiveComparer()).Compare( y, x ) );
      }
   }

   public static void Main()  {

      // Creates and initializes a new Array and a new custom comparer.
      String[] myKeys = { "red", "GREEN", "YELLOW", "BLUE", "purple",
"black", "orange" };
```

```
      String[] myValues = { "strawberries", "PEARS", "LIMES", "BERRIES",
"grapes", "olives", "cantaloupe" };
      IComparer myComparer = new myReverserClass();

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the default comparer.
      Array.Sort( myKeys, myValues, 1, 3 );
      Console.WriteLine( "After sorting a section of the Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, 1, 3, myComparer );
      Console.WriteLine( "After sorting a section of the Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the default comparer.
      Array.Sort( myKeys, myValues );
      Console.WriteLine( "After sorting the entire Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, myComparer );
      Console.WriteLine( "After sorting the entire Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );
   }

   public static void PrintKeysAndValues( String[] myKeys, String[]
myValues )  {
      for ( int i = 0; i < myKeys.Length; i++ )  {
         Console.WriteLine( "   {0,-10}: {1}", myKeys[i], myValues[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The Array initially contains the following values:
   red       : strawberries
   GREEN     : PEARS
   YELLOW    : LIMES
   BLUE      : BERRIES
   purple    : grapes
```

```
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the default comparer:
   red       : strawberries
   BLUE      : BERRIES
   GREEN     : PEARS
   YELLOW    : LIMES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the reverse case-insensitive
comparer:
   red       : strawberries
   YELLOW    : LIMES
   GREEN     : PEARS
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting the entire Array using the default comparer:
   black     : olives
   BLUE      : BERRIES
   GREEN     : PEARS
   orange    : cantaloupe
   purple    : grapes
   red       : strawberries
   YELLOW    : LIMES

After sorting the entire Array using the reverse case-insensitive
comparer:
   YELLOW    : LIMES
   red       : strawberries
   purple    : grapes
   orange    : cantaloupe
   GREEN     : PEARS
   BLUE      : BERRIES
   black     : olives

*/
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

If `comparer` is `null`, each key in the `keys` Array must implement the IComparable interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

.NET includes predefined IComparer implementations listed in the following table.

Expand table

Implementation	Description
System.Collections.CaseInsensitiveComparer	Compares any two objects, but performs a case-insensitive comparison of strings.
Comparer.Default	Compares any two objects by using the sorting conventions of the current culture.
Comparer.DefaultInvariant	Compares any two objects by using the sorting conventions of the invariant culture.
Comparer<T>.Default	Compares two objects of type `T` by using the type's default sort order.

You can also support custom comparisons by providing an instance of your own IComparer implementation to the `comparer` parameter. The example does this by defining an IComparer implementation that reverses the default sort order and performs case-insensitive string comparison.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `keys`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer
- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, Array)

Sorts a pair of one-dimensional Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable implementation of each key.

```
C#
```

```
public static void Sort (Array keys, Array? items);
```

Parameters

`keys` Array

The one-dimensional Array that contains the keys to sort.

`items` Array

The one-dimensional Array that contains the items that correspond to each of the keys in the `keys` Array.

-or-

`null` to sort only the `keys` Array.

Exceptions

ArgumentNullException

`keys` is `null`.

RankException

The `keys` Array is multidimensional.

-or-

The `items` Array is multidimensional.

ArgumentException

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

InvalidOperationException

One or more elements in the `keys` Array do not implement the IComparable interface.

Examples

The following example shows how to sort two associated arrays where the first array contains the keys and the second array contains the values. Sorts are done using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class SamplesArray  {

   public class myReverserClass : IComparer  {

      // Calls CaseInsensitiveComparer.Compare with the parameters
reversed.
      int IComparer.Compare( Object x, Object y )  {
          return( (new CaseInsensitiveComparer()).Compare( y, x ) );
      }
   }

   public static void Main()  {

      // Creates and initializes a new Array and a new custom comparer.
      String[] myKeys = { "red", "GREEN", "YELLOW", "BLUE", "purple",
"black", "orange" };
      String[] myValues = { "strawberries", "PEARS", "LIMES", "BERRIES",
"grapes", "olives", "cantaloupe" };
      IComparer myComparer = new myReverserClass();

      // Displays the values of the Array.
      Console.WriteLine( "The Array initially contains the following
values:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the default comparer.
      Array.Sort( myKeys, myValues, 1, 3 );
      Console.WriteLine( "After sorting a section of the Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts a section of the Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, 1, 3, myComparer );
      Console.WriteLine( "After sorting a section of the Array using the
reverse case-insensitive comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the default comparer.
      Array.Sort( myKeys, myValues );
      Console.WriteLine( "After sorting the entire Array using the
default comparer:" );
      PrintKeysAndValues( myKeys, myValues );

      // Sorts the entire Array using the reverse case-insensitive
comparer.
      Array.Sort( myKeys, myValues, myComparer );
      Console.WriteLine( "After sorting the entire Array using the
reverse case-insensitive comparer:" );
```

```
      PrintKeysAndValues( myKeys, myValues );
   }

   public static void PrintKeysAndValues( String[] myKeys, String[]
myValues )  {
      for ( int i = 0; i < myKeys.Length; i++ )  {
         Console.WriteLine( "   {0,-10}: {1}", myKeys[i], myValues[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The Array initially contains the following values:
   red       : strawberries
   GREEN     : PEARS
   YELLOW    : LIMES
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the default comparer:
   red       : strawberries
   BLUE      : BERRIES
   GREEN     : PEARS
   YELLOW    : LIMES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting a section of the Array using the reverse case-insensitive
comparer:
   red       : strawberries
   YELLOW    : LIMES
   GREEN     : PEARS
   BLUE      : BERRIES
   purple    : grapes
   black     : olives
   orange    : cantaloupe

After sorting the entire Array using the default comparer:
   black     : olives
   BLUE      : BERRIES
   GREEN     : PEARS
   orange    : cantaloupe
   purple    : grapes
   red       : strawberries
   YELLOW    : LIMES

After sorting the entire Array using the reverse case-insensitive
comparer:
```

```
    YELLOW     : LIMES
    red        : strawberries
    purple     : grapes
    orange     : cantaloupe
    GREEN      : PEARS
    BLUE       : BERRIES
    black      : olives

*/
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

Each key in the `keys` Array must implement the IComparable interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `keys`.

See also

- IComparable

- BinarySearch
- IDictionary
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array)

Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.

C#

```
public static void Sort (Array array);
```

Parameters

`array` Array

The one-dimensional Array to sort.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

InvalidOperationException

One or more elements in `array` do not implement the IComparable interface.

Examples

The following code example shows how to sort the values in an Array using the default comparer and a custom comparer that reverses the sort order. Note that the result might vary depending on the current CultureInfo.

C#

```
using System;
using System.Collections;

public class ReverseComparer : IComparer
{
   // Call CaseInsensitiveComparer.Compare with the parameters reversed.
   public int Compare(Object x, Object y)
   {
       return (new CaseInsensitiveComparer()).Compare(y, x );
   }
}

public class Example
{
   public static void Main()
   {
      // Create and initialize a new array.
      String[] words = { "The", "QUICK", "BROWN", "FOX", "jumps",
                         "over", "the", "lazy", "dog" };
      // Instantiate the reverse comparer.
      IComparer revComparer = new ReverseComparer();

      // Display the values of the array.
      Console.WriteLine( "The original order of elements in the array:"
);
      DisplayValues(words);

      // Sort a section of the array using the default comparer.
      Array.Sort(words, 1, 3);
      Console.WriteLine( "After sorting elements 1-3 by using the default
comparer:");
      DisplayValues(words);

      // Sort a section of the array using the reverse case-insensitive
comparer.
      Array.Sort(words, 1, 3, revComparer);
      Console.WriteLine( "After sorting elements 1-3 by using the reverse
case-insensitive comparer:");
      DisplayValues(words);
```

```
      // Sort the entire array using the default comparer.
      Array.Sort(words);
      Console.WriteLine( "After sorting the entire array by using the
default comparer:");
      DisplayValues(words);

      // Sort the entire array by using the reverse case-insensitive
comparer.
      Array.Sort(words, revComparer);
      Console.WriteLine( "After sorting the entire array using the
reverse case-insensitive comparer:");
      DisplayValues(words);
   }

   public static void DisplayValues(String[] arr)
   {
      for ( int i = arr.GetLowerBound(0); i <= arr.GetUpperBound(0);
            i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, arr[i] );
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//    The original order of elements in the array:
//       [0] : The
//       [1] : QUICK
//       [2] : BROWN
//       [3] : FOX
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the default comparer:
//       [0] : The
//       [1] : BROWN
//       [2] : FOX
//       [3] : QUICK
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the reverse case-insensitive
comparer:
//       [0] : The
//       [1] : QUICK
//       [2] : FOX
//       [3] : BROWN
//       [4] : jumps
//       [5] : over
//       [6] : the
```

```
//       [7] : lazy
//       [8] : dog
//
//    After sorting the entire array by using the default comparer:
//       [0] : BROWN
//       [1] : dog
//       [2] : FOX
//       [3] : jumps
//       [4] : lazy
//       [5] : over
//       [6] : QUICK
//       [7] : the
//       [8] : The
//
//    After sorting the entire array using the reverse case-insensitive
comparer:
//       [0] : the
//       [1] : The
//       [2] : QUICK
//       [3] : over
//       [4] : lazy
//       [5] : jumps
//       [6] : FOX
//       [7] : dog
//       [8] : BROWN
```

Remarks

Each element of `array` must implement the IComparable interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

See also

- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort(Array, IComparer)

Sorts the elements in a one-dimensional Array using the specified IComparer.

C#

```
public static void Sort (Array array, System.Collections.IComparer?
comparer);
```

Parameters

`array` Array

The one-dimensional array to sort.

`comparer` IComparer

The implementation to use when comparing elements.

-or-

`null` to use the IComparable implementation of each element.

Exceptions

ArgumentNullException

`array` is `null`.

RankException

`array` is multidimensional.

InvalidOperationException

`comparer` is `null`, and one or more elements in `array` do not implement the IComparable interface.

ArgumentException

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

Examples

The following example sorts the values in a string array by using the default comparer. It also defines a custom IComparer implementation named `ReverseComparer` that reverses an object's default sort order while performing a case-insensitive string comparison. Note that the output might vary depending on the current culture.

C#

```
using System;
using System.Collections;

public class ReverseComparer : IComparer
{
   // Call CaseInsensitiveComparer.Compare with the parameters reversed.
   public int Compare(Object x, Object y)
   {
       return (new CaseInsensitiveComparer()).Compare(y, x );
   }
}

public class Example
{
   public static void Main()
   {
      // Create and initialize a new array.
      String[] words = { "The", "QUICK", "BROWN", "FOX", "jumps",
                         "over", "the", "lazy", "dog" };
      // Instantiate the reverse comparer.
```

```
      IComparer revComparer = new ReverseComparer();

      // Display the values of the array.
      Console.WriteLine( "The original order of elements in the array:"
);
      DisplayValues(words);

      // Sort a section of the array using the default comparer.
      Array.Sort(words, 1, 3);
      Console.WriteLine( "After sorting elements 1-3 by using the default
comparer:");
      DisplayValues(words);

      // Sort a section of the array using the reverse case-insensitive
comparer.
      Array.Sort(words, 1, 3, revComparer);
      Console.WriteLine( "After sorting elements 1-3 by using the reverse
case-insensitive comparer:");
      DisplayValues(words);

      // Sort the entire array using the default comparer.
      Array.Sort(words);
      Console.WriteLine( "After sorting the entire array by using the
default comparer:");
      DisplayValues(words);

      // Sort the entire array by using the reverse case-insensitive
comparer.
      Array.Sort(words, revComparer);
      Console.WriteLine( "After sorting the entire array using the
reverse case-insensitive comparer:");
      DisplayValues(words);
   }

   public static void DisplayValues(String[] arr)
   {
      for ( int i = arr.GetLowerBound(0); i <= arr.GetUpperBound(0);
            i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, arr[i] );
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//    The original order of elements in the array:
//       [0] : The
//       [1] : QUICK
//       [2] : BROWN
//       [3] : FOX
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
```

```
//    After sorting elements 1-3 by using the default comparer:
//       [0] : The
//       [1] : BROWN
//       [2] : FOX
//       [3] : QUICK
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting elements 1-3 by using the reverse case-insensitive
comparer:
//       [0] : The
//       [1] : QUICK
//       [2] : FOX
//       [3] : BROWN
//       [4] : jumps
//       [5] : over
//       [6] : the
//       [7] : lazy
//       [8] : dog
//
//    After sorting the entire array by using the default comparer:
//       [0] : BROWN
//       [1] : dog
//       [2] : FOX
//       [3] : jumps
//       [4] : lazy
//       [5] : over
//       [6] : QUICK
//       [7] : the
//       [8] : The
//
//    After sorting the entire array using the reverse case-insensitive
comparer:
//       [0] : the
//       [1] : The
//       [2] : QUICK
//       [3] : over
//       [4] : lazy
//       [5] : jumps
//       [6] : FOX
//       [7] : dog
//       [8] : BROWN
```

Remarks

If `comparer` is `null`, each element of `array` must implement the IComparable interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

.NET includes predefined IComparer implementations listed in the following table.

Expand table

Implementation	Description
System.Collections.CaseInsensitiveComparer	Compares any two objects, but performs a case-insensitive comparison of strings.
Comparer.Default	Compares any two objects by using the sorting conventions of the current culture.
Comparer.DefaultInvariant	Compares any two objects by using the sorting conventions of the invariant culture.
Comparer<T>.Default	Compares two objects of type `T` by using the type's default sort order.

You can also support custom comparisons by providing an instance of your own IComparer implementation to the `comparer` parameter. The example does this by defining a `ReverseComparer` class that reverses the default sort order for instances of a type and performs case-insensitive string comparison.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception,

because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer
- IComparable
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<T>(T[])

Sorts the elements in an entire Array using the IComparable<T> generic interface implementation of each element of the Array.

C#

```
public static void Sort<T> (T[] array);
```

Type Parameters

T

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to sort.

Exceptions

ArgumentNullException

`array` is `null`.

InvalidOperationException

One or more elements in `array` do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<T>(T[]) generic method overload and the BinarySearch<T>(T[], T) generic method overload. An array of strings is created, in no particular order.

The array is displayed, sorted, and displayed again.

> **Note**
>
> The calls to the **Sort** and **BinarySearch** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

The BinarySearch<T>(T[], T) generic method overload is then used to search for two strings, one that is not in the array and one that is. The array and the return value of the BinarySearch method are passed to the `ShowWhere` generic method, which displays the index value if the string is found, and otherwise the elements the search string would fall between if it were in the array. The index is negative if the string is not n the array, so the `ShowWhere` method takes the bitwise complement (the ~ operator in C# and Visual C++, `Xor` -1 in Visual Basic) to obtain the index of the first element in the list that is larger than the search string.

```
C#
```

```
using System;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Tyrannosaurus",
                              "Mamenchisaurus",
                              "Deinonychus",
                              "Edmontosaurus"};

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nSort");
        Array.Sort(dinosaurs);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nBinarySearch for 'Coelophysis':");
        int index = Array.BinarySearch(dinosaurs, "Coelophysis");
        ShowWhere(dinosaurs, index);

        Console.WriteLine("\nBinarySearch for 'Tyrannosaurus':");
        index = Array.BinarySearch(dinosaurs, "Tyrannosaurus");
        ShowWhere(dinosaurs, index);
    }

    private static void ShowWhere<T>(T[] array, int index)
    {
        if (index<0)
        {
            // If the index is negative, it represents the bitwise
            // complement of the next larger element in the array.
            //
            index = ~index;

            Console.Write("Not found. Sorts between: ");

            if (index == 0)
                Console.Write("beginning of array and ");
            else
                Console.Write("{0} and ", array[index-1]);
```

```
            if (index == array.Length)
                Console.WriteLine("end of array.");
            else
                Console.WriteLine("{0}.", array[index]);
        }
        else
        {
            Console.WriteLine("Found at index {0}.", index);
        }
    }
}

/* This code example produces the following output:

Pachycephalosaurus
Amargasaurus
Tyrannosaurus
Mamenchisaurus
Deinonychus
Edmontosaurus

Sort

Amargasaurus
Deinonychus
Edmontosaurus
Mamenchisaurus
Pachycephalosaurus
Tyrannosaurus

BinarySearch for 'Coelophysis':
Not found. Sorts between: Amargasaurus and Deinonychus.

BinarySearch for 'Tyrannosaurus':
Found at index 5.
 */
```

Remarks

Each element of `array` must implement the IComparable<T> generic interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.

- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

See also

- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<T>(T[], IComparer<T>)

Sorts the elements in an Array using the specified IComparer<T> generic interface.

C#

```
public static void Sort<T> (T[] array,
System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-base Array to sort.

`comparer` IComparer<T>

The IComparer<T> generic interface implementation to use when comparing elements, or `null` to use the IComparable<T> generic interface implementation of each element.

Exceptions

ArgumentNullException

`array` is `null`.

InvalidOperationException

`comparer` is `null`, and one or more elements in `array` do not implement the IComparable<T> generic interface.

ArgumentException

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

Examples

The following code example demonstrates the Sort<T>(T[], IComparer<T>) generic method overload and the BinarySearch<T>(T[], T, IComparer<T>) generic method overload.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The array is displayed, sorted, and displayed again. Arrays must be sorted in order to use the BinarySearch method.

> **Note**
>
> The calls to the **Sort<T>(T[], IComparer<T>)** and **BinarySearch<T>(T[], T, IComparer<T>)** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

The BinarySearch<T>(T[], T, IComparer<T>) generic method overload is then used to search for two strings, one that is not in the array and one that is. The array and the return value of the BinarySearch<T>(T[], T, IComparer<T>) method are passed to the `ShowWhere` generic method, which displays the index value if the string is found, and otherwise the elements the search string would fall between if it were in the array. The index is negative if the string is not n the array, so the `ShowWhere` method takes the bitwise complement (the ~ operator in C# and Visual C++, `Xor` -1 in Visual Basic) to obtain the index of the first element in the list that is larger than the search string.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Tyrannosaurus",
                              "Mamenchisaurus",
                              "Deinonychus",
                              "Edmontosaurus"};

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
```

```
            Console.WriteLine(dinosaur);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort");
        Array.Sort(dinosaurs, rc);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nBinarySearch for 'Coelophysis':");
        int index = Array.BinarySearch(dinosaurs, "Coelophysis", rc);
        ShowWhere(dinosaurs, index);

        Console.WriteLine("\nBinarySearch for 'Tyrannosaurus':");
        index = Array.BinarySearch(dinosaurs, "Tyrannosaurus", rc);
        ShowWhere(dinosaurs, index);
    }

    private static void ShowWhere<T>(T[] array, int index)
    {
        if (index<0)
        {
            // If the index is negative, it represents the bitwise
            // complement of the next larger element in the array.
            //
            index = ~index;

            Console.Write("Not found. Sorts between: ");

            if (index == 0)
                Console.Write("beginning of array and ");
            else
                Console.Write("{0} and ", array[index-1]);

            if (index == array.Length)
                Console.WriteLine("end of array.");
            else
                Console.WriteLine("{0}.", array[index]);
        }
        else
        {
            Console.WriteLine("Found at index {0}.", index);
        }
    }
}

/* This code example produces the following output:

Pachycephalosaurus
Amargasaurus
```

```
Tyrannosaurus
Mamenchisaurus
Deinonychus
Edmontosaurus

Sort

Tyrannosaurus
Pachycephalosaurus
Mamenchisaurus
Edmontosaurus
Deinonychus
Amargasaurus

BinarySearch for 'Coelophysis':
Not found. Sorts between: Deinonychus and Amargasaurus.

BinarySearch for 'Tyrannosaurus':
Found at index 0.
 */
```

Remarks

If `comparer` is `null`, each element of `array` must implement the IComparable<T> generic interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^{N}, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer<T>
- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<T>(T[], Comparison<T>)

Sorts the elements in an Array using the specified Comparison<T>.

C#

```
public static void Sort<T> (T[] array, Comparison<T> comparison);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to sort.

`comparison` Comparison<T>

The Comparison<T> to use when comparing elements.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`comparison` is `null`.

ArgumentException

The implementation of `comparison` caused an error during the sort. For example, `comparison` might not return 0 when comparing an item with itself.

Examples

The following code example demonstrates the Sort(Comparison<T>) method overload.

The code example defines an alternative comparison method for strings, named `CompareDinosByLength`. This method works as follows: First, the comparands are tested for `null`, and a null reference is treated as less than a non-null. Second, the string lengths are compared, and the longer string is deemed to be greater. Third, if the lengths are equal, ordinary string comparison is used.

A array of strings is created and populated with four strings, in no particular order. The list also includes an empty string and a null reference. The list is displayed, sorted using a Comparison<T> generic delegate representing the `CompareDinosByLength` method, and displayed again.

```
C#
```

```
using System;
using System.Collections.Generic;

public class Example
{
    private static int CompareDinosByLength(string x, string y)
    {
        if (x == null)
        {
            if (y == null)
            {
                // If x is null and y is null, they're
                // equal.
                return 0;
            }
            else
            {
                // If x is null and y is not null, y
                // is greater.
                return -1;
            }
        }
        else
        {
            // If x is not null...
            //
            if (y == null)
                // ...and y is null, x is greater.
            {
                return 1;
            }
            else
            {
                // ...and y is not null, compare the
                // lengths of the two strings.
                //
                int retval = x.Length.CompareTo(y.Length);

                if (retval != 0)
                {
                    // If the strings are not of equal length,
                    // the longer string is greater.
                    //
                    return retval;
                }
                else
                {
                    // If the strings are of equal length,
                    // sort them with ordinary string comparison.
                    //
                    return x.CompareTo(y);
                }
            }
        }
```

```
    }

    public static void Main()
    {
        string[] dinosaurs = {
            "Pachycephalosaurus",
            "Amargasaurus",
            "",
            null,
            "Mamenchisaurus",
            "Deinonychus" };
        Display(dinosaurs);

        Console.WriteLine("\nSort with generic Comparison<string>
delegate:");
        Array.Sort(dinosaurs, CompareDinosByLength);
        Display(dinosaurs);
    }

    private static void Display(string[] arr)
    {
        Console.WriteLine();
        foreach( string s in arr )
        {
            if (s == null)
                Console.WriteLine("(null)");
            else
                Console.WriteLine("\"{0}\"", s);
        }
    }
}

/* This code example produces the following output:

"Pachycephalosaurus"
"Amargasaurus"
""
(null)
"Mamenchisaurus"
"Deinonychus"

Sort with generic Comparison<string> delegate:

(null)
""
"Deinonychus"
"Amargasaurus"
"Mamenchisaurus"
"Pachycephalosaurus"
 */
```

Remarks

If the sort is not successfully completed, the results are undefined.

This method uses introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 6 elements.

See also

- Comparison<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<T>(T[], Int32, Int32)

Sorts the elements in a range of elements in an Array using the IComparable<T> generic interface implementation of each element of the Array.

```
C#

public static void Sort<T> (T[] array, int index, int length);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to sort.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

InvalidOperationException

One or more elements in `array` do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<T>(T[], Int32, Int32) generic method overload and the Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overload for sorting a range in an array.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names, consisting of three herbivores followed by three carnivores (tyrannosaurids, to be precise). The Sort<T>(T[], Int32, Int32) generic method overload is used to sort the last three elements of the array, which is then displayed. The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overload is used with `ReverseCompare` to sort the last three elements in reverse order. The thoroughly confused dinosaurs are displayed again.

> **Note**
>
> The calls to the **Sort<T>(T[], IComparer<T>)** and **BinarySearch<T>(T[], T, IComparer<T>)** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

```
C#
```

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Mamenchisaurus",
                              "Tarbosaurus",
                              "Tyrannosaurus",
                              "Albertasaurus"};

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nSort(dinosaurs, 3, 3)");
        Array.Sort(dinosaurs, 3, 3);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort(dinosaurs, 3, 3, rc)");
        Array.Sort(dinosaurs, 3, 3, rc);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }
    }
}

/* This code example produces the following output:

Pachycephalosaurus
```

```
Amargasaurus
Mamenchisaurus
Tarbosaurus
Tyrannosaurus
Albertasaurus

Sort(dinosaurs, 3, 3)

Pachycephalosaurus
Amargasaurus
Mamenchisaurus
Albertasaurus
Tarbosaurus
Tyrannosaurus

Sort(dinosaurs, 3, 3, rc)

Pachycephalosaurus
Amargasaurus
Mamenchisaurus
Tyrannosaurus
Tarbosaurus
Albertasaurus
 */
```

Remarks

Each element within the specified range of elements in `array` must implement the IComparable<T> generic interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

See also

- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<T>(T[], Int32, Int32, IComparer<T>)

Sorts the elements in a range of elements in an Array using the specified IComparer<T> generic interface.

C#

```
public static void Sort<T> (T[] array, int index, int length,
System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to sort.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

`comparer` IComparer<T>

The IComparer<T> generic interface implementation to use when comparing elements, or `null` to use the IComparable<T> generic interface implementation of each element.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `array`.

-or-

`length` is less than zero.

ArgumentException

`index` and `length` do not specify a valid range in `array`.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException

`comparer` is `null`, and one or more elements in `array` do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<T>(T[], Int32, Int32) generic method overload and the Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overload for sorting a range in an array.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in

Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names, consisting of three herbivores followed by three carnivores (tyrannosaurids, to be precise). The Sort<T>(T[], Int32, Int32) generic method overload is used to sort the last three elements of the array, which is then displayed. The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overload is used with `ReverseCompare` to sort the last three elements in reverse order. The thoroughly confused dinosaurs are displayed again.

> **Note**
>
> The calls to the **Sort<T>(T[], IComparer<T>)** and **BinarySearch<T>(T[], T, IComparer<T>)** generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first argument. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {"Pachycephalosaurus",
                              "Amargasaurus",
                              "Mamenchisaurus",
                              "Tarbosaurus",
                              "Tyrannosaurus",
                              "Albertasaurus"};

        Console.WriteLine();
```

```
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nSort(dinosaurs, 3, 3)");
        Array.Sort(dinosaurs, 3, 3);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort(dinosaurs, 3, 3, rc)");
        Array.Sort(dinosaurs, 3, 3, rc);

        Console.WriteLine();
        foreach( string dinosaur in dinosaurs )
        {
            Console.WriteLine(dinosaur);
        }
    }
}

/* This code example produces the following output:

Pachycephalosaurus
Amargasaurus
Mamenchisaurus
Tarbosaurus
Tyrannosaurus
Albertasaurus

Sort(dinosaurs, 3, 3)

Pachycephalosaurus
Amargasaurus
Mamenchisaurus
Albertasaurus
Tarbosaurus
Tyrannosaurus

Sort(dinosaurs, 3, 3, rc)

Pachycephalosaurus
Amargasaurus
Mamenchisaurus
Tyrannosaurus
Tarbosaurus
Albertasaurus
 */
```

Remarks

If `comparer` is `null`, each element within the specified range of elements in `array` must implement the IComparable<T> generic interface to be capable of comparisons with every other element in `array`.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^{N}, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer<T>
- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>)

Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.

C#

```
public static void Sort<TKey,TValue> (TKey[] keys, TValue[]? items, int
index, int length, System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TKey`

The type of the elements of the key array.

`TValue`

The type of the elements of the items array.

Parameters

`keys` TKey[]

The one-dimensional, zero-based Array that contains the keys to sort.

`items` TValue[]

The one-dimensional, zero-based Array that contains the items that correspond to the keys in `keys`, or `null` to sort only `keys`.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

`comparer` IComparer<TKey>

The IComparer<T> generic interface implementation to use when comparing elements, or `null` to use the IComparable<T> generic interface implementation of each element.

Exceptions

ArgumentNullException

`keys` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `keys`.

-or-

`length` is less than zero.

ArgumentException

`items` is not `null`, and the lower bound of `keys` does not match the lower bound of `items`.

-or-

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

`index` and `length` do not specify a valid range in the `keys` Array.

-or-

`items` is not `null`, and `index` and `length` do not specify a valid range in the `items` Array.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException
`comparer` is `null`, and one or more elements in the `keys` Array do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<TKey,TValue>(TKey[], TValue[]), Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>), Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32), and Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overloads, for sorting pairs of arrays that represent keys and values.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names (the keys) and an array of integers representing the maximum length of each dinosaur in meters (the values). The arrays are then sorted and displayed several times:

- The Sort<TKey,TValue>(TKey[], TValue[]) overload is used to sort both arrays in order of the dinosaur names in the first array.
- The Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) overload and an instance of `ReverseCompare` are used to reverse the sort order of the paired arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32) overload is used to sort the last three elements of both arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) overload is used to sort the last three elements of both arrays in reverse order.

Note

The calls to the generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first two arguments. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {
            "Seismosaurus",
            "Chasmosaurus",
            "Coelophysis",
            "Mamenchisaurus",
            "Caudipteryx",
            "Cetiosaurus"  };

        int[] dinosaurSizes = { 40, 5, 3, 22, 1, 18 };

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes)");
        Array.Sort(dinosaurs, dinosaurSizes);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        ReverseComparer rc = new ReverseComparer();
```

```
        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }
    }
}

/* This code example produces the following output:

Seismosaurus: up to 40 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.

Sort(dinosaurs, dinosaurSizes)

Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Seismosaurus: up to 40 meters long.

Sort(dinosaurs, dinosaurSizes, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
```

```
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.
 */
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

If `comparer` is `null`, each key within the specified range of elements in the `keys` Array must implement the IComparable<T> generic interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.

- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer<T>
- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<TKey,TValue>(TKey[], TValue[])

Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable<T> generic interface implementation of each key.

```
C#
public static void Sort<TKey,TValue> (TKey[] keys, TValue[]? items);
```

Type Parameters

`TKey`

The type of the elements of the key array.

`TValue`

The type of the elements of the items array.

Parameters

`keys` TKey[]

The one-dimensional, zero-based Array that contains the keys to sort.

`items` TValue[]

The one-dimensional, zero-based Array that contains the items that correspond to the keys in `keys`, or `null` to sort only `keys`.

Exceptions

ArgumentNullException

`keys` is `null`.

ArgumentException

`items` is not `null`, and the lower bound of `keys` does not match the lower bound of `items`.

-or-

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

InvalidOperationException

One or more elements in the `keys` Array do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<TKey,TValue>(TKey[], TValue[]), Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>), Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32), and Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overloads, for sorting pairs of arrays that represent keys and values.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names (the keys) and an array of integers representing the maximum length of each dinosaur in meters (the values). The arrays are then sorted and displayed several times:

- The Sort<TKey,TValue>(TKey[], TValue[]) overload is used to sort both arrays in order of the dinosaur names in the first array.
- The Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) overload and an instance of `ReverseCompare` are used to reverse the sort order of the paired arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32) overload is used to sort the last three elements of both arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) overload is used to sort the last three elements of both arrays in reverse order.

> **Note**
>
> The calls to the generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first two arguments. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {
            "Seismosaurus",
            "Chasmosaurus",
            "Coelophysis",
            "Mamenchisaurus",
            "Caudipteryx",
            "Cetiosaurus"  };

        int[] dinosaurSizes = { 40, 5, 3, 22, 1, 18 };

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes)");
        Array.Sort(dinosaurs, dinosaurSizes);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
```

```
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }
    }
}

/* This code example produces the following output:

Seismosaurus: up to 40 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.

Sort(dinosaurs, dinosaurSizes)

Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Seismosaurus: up to 40 meters long.

Sort(dinosaurs, dinosaurSizes, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3)

Seismosaurus: up to 40 meters long.
```

```
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.
 */
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

Each key in the `keys` Array must implement the IComparable<T> generic interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

See also

- IComparable<T>
- BinarySearch
- IDictionary<TKey,TValue>
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>)

Sorts a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the specified IComparer<T> generic interface.

C#

```
public static void Sort<TKey,TValue> (TKey[] keys, TValue[]? items,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

TKey

The type of the elements of the key array.

`TValue`

The type of the elements of the items array.

Parameters

`keys` TKey[]

The one-dimensional, zero-based Array that contains the keys to sort.

`items` TValue[]

The one-dimensional, zero-based Array that contains the items that correspond to the keys in `keys`, or `null` to sort only `keys`.

`comparer` IComparer<TKey>

The IComparer<T> generic interface implementation to use when comparing elements, or `null` to use the IComparable<T> generic interface implementation of each element.

Exceptions

ArgumentNullException

`keys` is `null`.

ArgumentException

`items` is not `null`, and the lower bound of `keys` does not match the lower bound of `items`.

-or-

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

The implementation of `comparer` caused an error during the sort. For example, `comparer` might not return 0 when comparing an item with itself.

InvalidOperationException

`comparer` is `null`, and one or more elements in the `keys` Array do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<TKey,TValue>(TKey[], TValue[]), [], Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>), Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32), and Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overloads, for sorting pairs of arrays that represent keys and values.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names (the keys) and an array of integers representing the maximum length of each dinosaur in meters (the values). The arrays are then sorted and displayed several times:

- The Sort<TKey,TValue>(TKey[], TValue[]) overload is used to sort both arrays in order of the dinosaur names in the first array.
- The Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) overload and an instance of `ReverseCompare` are used to reverse the sort order of the paired arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32) overload is used to sort the last three elements of both arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) overload is used to sort the last three elements of both arrays in reverse order.

> **Note**
>
> The calls to the generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first two arguments. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
```

```
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {
            "Seismosaurus",
            "Chasmosaurus",
            "Coelophysis",
            "Mamenchisaurus",
            "Caudipteryx",
            "Cetiosaurus"  };

        int[] dinosaurSizes = { 40, 5, 3, 22, 1, 18 };

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes)");
        Array.Sort(dinosaurs, dinosaurSizes);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
```

```
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }
    }
}

/* This code example produces the following output:

Seismosaurus: up to 40 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.

Sort(dinosaurs, dinosaurSizes)

Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Seismosaurus: up to 40 meters long.

Sort(dinosaurs, dinosaurSizes, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3, rc)
```

```
 Seismosaurus: up to 40 meters long.
 Mamenchisaurus: up to 22 meters long.
 Coelophysis: up to 3 meters long.
 Chasmosaurus: up to 5 meters long.
 Cetiosaurus: up to 18 meters long.
 Caudipteryx: up to 1 meters long.
  */
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

If `comparer` is `null`, each key in the `keys` Array must implement the IComparable<T> generic interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is the Length of `array`.

Notes to Callers

.NET Framework 4 and earlier versions used only the Quicksort algorithm. Quicksort identifies invalid comparers in some situations in which the sorting operation throws an IndexOutOfRangeException exception, and throws an ArgumentException exception to the caller. Starting with .NET Framework 4.5, it is possible that sorting operations that previously threw ArgumentException will not throw an exception, because the insertion sort and heapsort algorithms do not detect an invalid comparer. For the most part, this applies to arrays with less than or equal to 16 elements.

See also

- IComparer<T>
- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32)

Sorts a range of elements in a pair of Array objects (one contains the keys and the other contains the corresponding items) based on the keys in the first Array using the IComparable<T> generic interface implementation of each key.

```
C#
```

```
public static void Sort<TKey,TValue> (TKey[] keys, TValue[]? items, int
index, int length);
```

Type Parameters

`TKey`

The type of the elements of the key array.

`TValue`

The type of the elements of the items array.

Parameters

`keys` TKey[]

The one-dimensional, zero-based Array that contains the keys to sort.

`items` TValue[]

The one-dimensional, zero-based Array that contains the items that correspond to the keys in `keys`, or `null` to sort only `keys`.

`index` Int32

The starting index of the range to sort.

`length` Int32

The number of elements in the range to sort.

Exceptions

ArgumentNullException

`keys` is `null`.

ArgumentOutOfRangeException

`index` is less than the lower bound of `keys`.

-or-

`length` is less than zero.

ArgumentException

`items` is not `null`, and the lower bound of `keys` does not match the lower bound of `items`.

-or-

`items` is not `null`, and the length of `keys` is greater than the length of `items`.

-or-

`index` and `length` do not specify a valid range in the `keys` Array.

-or-

`items` is not `null`, and `index` and `length` do not specify a valid range in the `items` Array.

InvalidOperationException
One or more elements in the `keys` Array do not implement the IComparable<T> generic interface.

Examples

The following code example demonstrates the Sort<TKey,TValue>(TKey[], TValue[]), Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>), Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32), and Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) generic method overloads, for sorting pairs of arrays that represent keys and values.

The code example defines an alternative comparer for strings, named `ReverseCompare`, which implements the `IComparer<string>` (`IComparer(Of String)` in Visual Basic, `IComparer<String^>` in Visual C++) generic interface. The comparer calls the CompareTo(String) method, reversing the order of the comparands so that the strings sort high-to-low instead of low-to-high.

The code example creates and displays an array of dinosaur names (the keys) and an array of integers representing the maximum length of each dinosaur in meters (the values). The arrays are then sorted and displayed several times:

- The Sort<TKey,TValue>(TKey[], TValue[]) overload is used to sort both arrays in order of the dinosaur names in the first array.
- The Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) overload and an instance of `ReverseCompare` are used to reverse the sort order of the paired

arrays.

- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32) overload is used to sort the last three elements of both arrays.
- The Sort<TKey,TValue>(TKey[], TValue[], Int32, Int32, IComparer<TKey>) overload is used to sort the last three elements of both arrays in reverse order.

> **Note**
>
> The calls to the generic methods do not look any different from calls to their nongeneric counterparts, because Visual Basic, C#, and C++ infer the type of the generic type parameter from the type of the first two arguments. If you use the **Ildasm.exe (IL Disassembler)** to examine the Microsoft intermediate language (MSIL), you can see that the generic methods are being called.

C#

```
using System;
using System.Collections.Generic;

public class ReverseComparer: IComparer<string>
{
    public int Compare(string x, string y)
    {
        // Compare y and x in reverse order.
        return y.CompareTo(x);
    }
}

public class Example
{
    public static void Main()
    {
        string[] dinosaurs = {
            "Seismosaurus",
            "Chasmosaurus",
            "Coelophysis",
            "Mamenchisaurus",
            "Caudipteryx",
            "Cetiosaurus"  };

        int[] dinosaurSizes = { 40, 5, 3, 22, 1, 18 };

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
```

```
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes)");
        Array.Sort(dinosaurs, dinosaurSizes);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        ReverseComparer rc = new ReverseComparer();

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }

        Console.WriteLine("\nSort(dinosaurs, dinosaurSizes, 3, 3, rc)");
        Array.Sort(dinosaurs, dinosaurSizes, 3, 3, rc);

        Console.WriteLine();
        for (int i = 0; i < dinosaurs.Length; i++)
        {
            Console.WriteLine("{0}: up to {1} meters long.",
                dinosaurs[i], dinosaurSizes[i]);
        }
    }
}

/* This code example produces the following output:

Seismosaurus: up to 40 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
```

```
Sort(dinosaurs, dinosaurSizes)

Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.
Coelophysis: up to 3 meters long.
Mamenchisaurus: up to 22 meters long.
Seismosaurus: up to 40 meters long.

Sort(dinosaurs, dinosaurSizes, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Caudipteryx: up to 1 meters long.
Cetiosaurus: up to 18 meters long.
Chasmosaurus: up to 5 meters long.

Sort(dinosaurs, dinosaurSizes, 3, 3, rc)

Seismosaurus: up to 40 meters long.
Mamenchisaurus: up to 22 meters long.
Coelophysis: up to 3 meters long.
Chasmosaurus: up to 5 meters long.
Cetiosaurus: up to 18 meters long.
Caudipteryx: up to 1 meters long.
 */
```

Remarks

Each key in the `keys` Array has a corresponding item in the `items` Array. When a key is repositioned during the sorting, the corresponding item in the `items` Array is similarly repositioned. Therefore, the `items` Array is sorted according to the arrangement of the corresponding keys in the `keys` Array.

Each key within the specified range of elements in the `keys` Array must implement the IComparable<T> generic interface to be capable of comparisons with every other key.

You can sort if there are more items than keys, but the items that have no corresponding keys will not be sorted. You cannot sort if there are more keys than items; doing this throws an ArgumentException.

If the sort is not successfully completed, the results are undefined.

This method uses the introspective sort (introsort) algorithm as follows:

- If the partition size is less than or equal to 16 elements, it uses an insertion sort algorithm.
- If the number of partitions exceeds 2 * Log^N, where *N* is the range of the input array, it uses a Heapsort algorithm.
- Otherwise, it uses a Quicksort algorithm.

This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

This method is an O(`n` log `n`) operation, where `n` is `length`.

See also

- IComparable<T>
- BinarySearch
- Performing Culture-Insensitive String Operations in Arrays

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.TrueForAll<T>(T[], Predicate<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether every element in the array matches the conditions defined by the specified predicate.

```
C#

public static bool TrueForAll<T> (T[] array, Predicate<T> match);
```

Type Parameters

`T`

The type of the elements of the array.

Parameters

`array` T[]

The one-dimensional, zero-based Array to check against the conditions.

`match` Predicate<T>

The predicate that defines the conditions to check against the elements.

Returns

Boolean

`true` if every element in `array` matches the conditions defined by the specified predicate; otherwise, `false`. If there are no elements in the array, the return value is `true`.

Exceptions

ArgumentNullException

`array` is `null`.

-or-

`match` is `null`.

Examples

The following example determines whether the last character of each element in a string array is a number. It creates two string arrays. The first array includes both strings that end with alphabetic characters and strings that end with numeric characters. The second array consists only of strings that end with numeric characters. The example also defines an `EndWithANumber` method whose signature matches the Predicate<T> delegate. The example passes each array to the TrueForAll method along with a delegate that represents the `EndsWithANumber` method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] values1 = { "Y2K", "A2000", "DC2A6", "MMXIV", "0C3" };
      String[] values2 = { "Y2", "A2000", "DC2A6", "MMXIV_0", "0C3" };

      if (Array.TrueForAll(values1, EndsWithANumber))
         Console.WriteLine("All elements end with an integer.");
      else
         Console.WriteLine("Not all elements end with an integer.");

      if (Array.TrueForAll(values2, EndsWithANumber))
         Console.WriteLine("All elements end with an integer.");
      else
         Console.WriteLine("Not all elements end with an integer.");
   }

   private static bool EndsWithANumber(string value)
   {
      int s;
      return int.TryParse(value.Substring(value.Length - 1), out s);
   }
}
// The example displays the following output:
//       Not all elements end with an integer.
//       All elements end with an integer.
```

The following example is similar to the first, except that it passes the string array to the TrueForAll method along with a lambda expression that determines whether a particular array element ends with the string representation of a number.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] values = { "Y2K", "A2000", "DC2A6", "MMXIV", "0C3" };
      if (Array.TrueForAll(values, value => {
                                      int s;
                                      return
int.TryParse(value.Substring(value.Length - 1), out s); }
                                    ))
         Console.WriteLine("All elements end with an integer.");
      else
         Console.WriteLine("Not all elements end with an integer.");
   }
}
// The example displays the following output:
//        Not all elements end with an integer.
```

In both cases, the TrueForAll method returns `false` as soon as it encounters the first array element that does not end in a number. Otherwise, it returns `true` after iterating all the elements in the array.

> **Note**
>
> As both examples show, in C# and Visual Basic, it is not necessary to create the `Predicate<string>` delegate (`Predicate(Of String)` in Visual Basic) explicitly. These languages infer the correct delegate from context and create it automatically.

Remarks

The Predicate<T> is a delegate to a method that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of `array` are individually passed to the Predicate<T>, and processing is stopped when the delegate returns `false` for any element.

This method is an O(`n`) operation, where `n` is the Length of `array`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exists<T>(T[], Predicate<T>)
- Predicate<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.ICollection.Count Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements contained in the Array.

C#

```
int System.Collections.ICollection.Count { get; }
```

Property Value

Int32

The number of elements contained in the collection.

Implements

Count

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an ICollection interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.IList.Add(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calling this method always throws a NotSupportedException exception.

```
C#
int IList.Add (object value);
```

Parameters

`value` Object

The object to be added to the IList.

Returns

Int32

Adding a value to an array is not supported. No value is returned.

Implements

Add(Object)

Exceptions

NotSupportedException

The IList has a fixed size.

Remarks

Ordinarily, an IList.Add implementation adds a member to a collection. However, because arrays have a fixed size (the IsFixedSize property always returns `true`), this method always throws a NotSupportedException exception.

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IList.Clear Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes all items from the IList.

```
C#
void IList.Clear ();
```

Implements

Clear()

Exceptions

NotSupportedException

The IList is read-only.

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.IList.Contains(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether an element is in the IList.

```
C#
bool IList.Contains (object value);
```

Parameters

`value` Object

The object to locate in the current list. The element to locate can be `null` for reference types.

Returns

Boolean

`true` if `value` is found in the IList; otherwise, `false`.

Implements

Contains(Object)

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.IList.IndexOf(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines the index of a specific item in the IList.

```
C#
int IList.IndexOf (object value);
```

Parameters

`value` Object

The object to locate in the current list.

Returns

Int32

The index of value if found in the list; otherwise, -1.

Implements

IndexOf(Object)

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array.IList.Insert(Int32, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Inserts an item to the IList at the specified index.

```
C#
void IList.Insert (int index, object value);
```

Parameters

index Int32

The index at which `value` should be inserted.

value Object

The object to insert.

Implements

Insert(Int32, Object)

Exceptions

ArgumentOutOfRangeException

`index` is not a valid index in the IList.

NotSupportedException

The IList is read-only.

-or-

The IList has a fixed size.

NullReferenceException

`value` is null reference in the IList.

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IList.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the element at the specified index.

```
C#
object? System.Collections.IList.Item[int index] { get; set; }
```

Parameters

`index` Int32

The index of the element to get or set.

Property Value

Object

The element at the specified index.

Implements

Item[Int32]

Exceptions

ArgumentOutOfRangeException

`index` is less than zero.

-or-

`index` is equal to or greater than Count.

ArgumentException

The current Array does not have exactly one dimension.

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IList.Remove(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes the first occurrence of a specific object from the IList.

C#

```
void IList.Remove (object value);
```

Parameters

`value` Object

The object to remove from the IList.

Implements

Remove(Object)

Exceptions

NotSupportedException

The IList is read-only.

-or-

The IList has a fixed size.

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IList.RemoveAt(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes the IList item at the specified index.

```
C#

void IList.RemoveAt (int index);
```

Parameters

`index` Int32

The index of the element to remove.

Implements

RemoveAt(Int32)

Exceptions

ArgumentOutOfRangeException

index is not a valid index in the IList.

NotSupportedException

The IList is read-only.

-or-

The IList has a fixed size.

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IList interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IStructuralComparable.CompareTo(Object, IComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the current collection object precedes, occurs in the same position as, or follows another object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IComparer

An object that compares the current object and `other`.

Returns

Int32

An integer that indicates the relationship of the current collection object to other, as shown in the following table.

Expand table

Return value	Description
-1	The current instance precedes `other`.
0	The current instance and `other` are equal.
1	The current instance follows `other`.

Implements

CompareTo(Object, IComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IStructuralComparable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IStructuralEquatable.Equals(Object, IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether an object is equal to the current instance.

```
C#

bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IEqualityComparer

An object that determines whether the current instance and `other` are equal.

Returns

Boolean

`true` if the two objects are equal; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IStructuralEquatable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array.IStructuralEquatable.GetHashCode(IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a hash code for the current instance.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object that computes the hash code of the current object.

Returns

Int32

The hash code for the current instance.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Array instance is cast to an IStructuralEquatable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.Enumerator Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an enumerator for the elements of an ArraySegment<T>.

C#

```
public struct ArraySegment<T>.Enumerator :
System.Collections.Generic.IEnumerator<T>
```

Type Parameters

T

Inheritance Object → ValueType → ArraySegment<T>.Enumerator

Implements IEnumerator<T> , IEnumerator , IDisposable

Remarks

The C# foreach of the C# language and the For Each...Next construct in Visual Basic hides the complexity of enumerators. Instead of directly manipulating the enumerator, using `foreach` or `For Each...Next` is recommended.

Initially, the enumerator is positioned before the first element in the ArraySegment<T>. At this position, Current is undefined. You must call MoveNext to advance the enumerator to the first item in the ArraySegment<T> before reading the value of Current.

Current returns the same value until MoveNext is called. MoveNext sets Current to the next item in the ArraySegment<T>.

If MoveNext passes the end of the ArraySegment<T>, MoveNext returns `false`. When the enumerator is at this state, subsequent calls to MoveNext also return `false` and

Current is undefined. You cannot set Current to the first item in the ArraySegment<T> again; you must create a new enumerator instance instead.

Properties

Expand table

Current	Gets a reference to the item at the current position of the enumerator.

Methods

Expand table

Dispose()	Releases all resources used by this enumerator instance.
MoveNext()	Advances the enumerator to the next element of the ArraySegment<T>.

Explicit Interface Implementations

Expand table

IEnumerator.Current	Gets the element at the current position of the enumerator.
IEnumerator.Reset()	Sets the enumerator to its initial position, which is before the first element in the array segment.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ArraySegment<T>.Enumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a reference to the item at the current position of the enumerator.

```
C#
public T Current { get; }
```

Property Value

T

The element of the ArraySegment<T> at the current position of the enumerator.

Implements

Current

Remarks

`Current` is undefined under either of the following conditions:

- Immediately after the enumerator is created, the enumerator is positioned before the first element in the array segment. MoveNext must be called to advance the enumerator to the first element of the array segment before reading the value of `Current`.
- The last call to MoveNext returned `false`, which indicates the end of the array segment.

`Current` returns the same value until MoveNext is called. MoveNext sets `Current` to the next item in the array segment.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Enumerator.Dispose Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Releases all resources used by this enumerator instance.

C#

```
public void Dispose ();
```

Implements

Dispose()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Enumerator.MoveNext Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Advances the enumerator to the next element of the ArraySegment<T>.

```
C#
public bool MoveNext ();
```

Returns

Boolean

`true` if the enumerator was successfully advanced to the next element; `false` if the enumerator has passed the end of the array segment.

Implements

MoveNext()

Remarks

After an enumerator is created, it is positioned before the first element in the array segment, and the first call to `MoveNext` advances the enumerator to the first item in the array segment.

If `MoveNext` passes the end of the array segment, `MoveNext` returns `false`. When the enumerator is at this state, subsequent calls to `MoveNext` also return `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Enumerator.IEnumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the element at the current position of the enumerator.

```
C#
object? System.Collections.IEnumerator.Current { get; }
```

Property Value

Object

The element in the array segment at the current position of the enumerator.

Implements

Current

Exceptions

InvalidOperationException

The enumerator is positioned before the first element of the array segment or after the last element.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T>.Enumerator instance is cast to an IEnumerator interface.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Enumerator.IEnumerator.Reset Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the enumerator to its initial position, which is before the first element in the array segment.

C#

```
void IEnumerator.Reset ();
```

Implements

Reset()

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T>.Enumerator instance is cast to an IEnumerator interface.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ArraySegment<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Delimits a section of a one-dimensional array.

C#

```
public readonly struct ArraySegment<T> :
System.Collections.Generic.ICollection<T>,
System.Collections.Generic.IEnumerable<T>,
System.Collections.Generic.IList<T>,
System.Collections.Generic.IReadOnlyCollection<T>,
System.Collections.Generic.IReadOnlyList<T>
```

Type Parameters

T

The type of the elements in the array segment.

Inheritance Object → ValueType → ArraySegment<T>

Implements ICollection<T> , IEnumerable<T> , IList<T> , IReadOnlyCollection<T> , IReadOnlyList<T> , IEnumerable

Examples

The following code example passes an ArraySegment<T> structure to a method.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
```

```
      String[] myArr = { "The", "quick", "brown", "fox", "jumps", "over",
"the", "lazy", "dog" };

      // Display the initial contents of the array.
      Console.WriteLine( "The original array initially contains:" );
      PrintIndexAndValues( myArr );

      // Define an array segment that contains the entire array.
      ArraySegment<string> myArrSegAll = new ArraySegment<string>( myArr );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The first array segment (with all the array's
elements) contains:" );
      PrintIndexAndValues( myArrSegAll );

      // Define an array segment that contains the middle five values of the
array.
      ArraySegment<string> myArrSegMid = new ArraySegment<string>( myArr, 2,
5 );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The second array segment (with the middle five
elements) contains:" );
      PrintIndexAndValues( myArrSegMid );

      // Modify the fourth element of the first array segment myArrSegAll.
      myArrSegAll.Array[3] = "LION";

      // Display the contents of the second array segment myArrSegMid.
      // Note that the value of its second element also changed.
      Console.WriteLine( "After the first array segment is modified, the
second array segment now contains:" );
      PrintIndexAndValues( myArrSegMid );
   }

   public static void PrintIndexAndValues( ArraySegment<string> arrSeg )  {
      for ( int i = arrSeg.Offset; i < (arrSeg.Offset + arrSeg.Count); i++ )
{
         Console.WriteLine( "   [{0}] : {1}", i, arrSeg.Array[i] );
      }
      Console.WriteLine();
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.
```

```
The original array initially contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The first array segment (with all the array's elements) contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The second array segment (with the middle five elements) contains:
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the

After the first array segment is modified, the second array segment now
contains:
   [2] : brown
   [3] : LION
   [4] : jumps
   [5] : over
   [6] : the

*/
```

Remarks

ArraySegment<T> is a wrapper around an array that delimits a range of elements in that array. Multiple ArraySegment<T> instances can refer to the same original array and can overlap. The original array must be one-dimensional and must have zero-based indexing.

> ⓘ **Note**

ArraySegment<T> implements the **IReadOnlyCollection<T>** interface starting with the .NET Framework 4.6; in previous versions of the .NET Framework, the **ArraySegment<T>** structure did not implement this interface.

The ArraySegment<T> structure is useful whenever the elements of an array will be manipulated in distinct segments. For example:

- You can pass an ArraySegment<T> object that represents only a portion of an array as an argument to a method, rather than call a relatively expensive method like Copy to pass a copy of a portion of an array.
- In a multithreaded app, you can use the ArraySegment<T> structure to have each thread operate on only a portion of the array.
- For task-based asynchronous operations, you can use an ArraySegment<T> object to ensure that each task operates on a distinct segment of the array. The following example divides an array into individual segments with up to ten elements. Each element in the segment is multiplied by its segment number. The result shows that using the ArraySegment<T> class to manipulate elements in this way changes the values of its underlying array.

C#

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

public class Example
{
   private const int segmentSize = 10;

   public static async Task Main()
   {
      List<Task> tasks = new List<Task>();

      // Create array.
      int[] arr = new int[50];
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
         arr[ctr] = ctr + 1;

      // Handle array in segments of 10.
      for (int ctr = 1; ctr <=
Math.Ceiling(((double)arr.Length)/segmentSize); ctr++) {
         int multiplier = ctr;
         int elements = (multiplier - 1) * 10 + segmentSize >
arr.Length ?
                           arr.Length - (multiplier - 1) * 10 :
segmentSize;
```

```
         ArraySegment<int> segment = new ArraySegment<int>(arr, (ctr -
1) * 10, elements);
         tasks.Add(Task.Run( () => { IList<int> list = (IList<int>)
segment;
                                     for (int index = 0; index <
list.Count; index++)
                                        list[index] = list[index] *
multiplier;
                                  } ));
      }
      try {
         await Task.WhenAll(tasks.ToArray());
         int elementsShown = 0;
         foreach (var value in arr) {
            Console.Write("{0,3} ", value);
            elementsShown++;
            if (elementsShown % 18 == 0)
               Console.WriteLine();
         }
      }
      catch (AggregateException e) {
         Console.WriteLine("Errors occurred when working with the
array:");
         foreach (var inner in e.InnerExceptions)
            Console.WriteLine("{0}: {1}", inner.GetType().Name,
inner.Message);
      }
   }
}
// The example displays the following output:
//      1   2   3   4   5   6   7   8   9  10  22  24  26  28  30  32
34  36
//     38  40  63  66  69  72  75  78  81  84  87  90 124 128 132 136
140 144
//    148 152 156 160 205 210 215 220 225 230 235 240 245 250
```

Note, however, that although the ArraySegment<T> structure can be used to divide an array into distinct segments, the segments are not completely independent of one another. The Array property returns the entire original array, not a copy of the array; therefore, changes made to the array returned by the Array property are made to the original array. If this is undesirable, you should perform operations on a copy of the array, rather than an ArraySegment<T> object that represents a portion of the array.

The Equals method and the equality and inequality operators test for reference equality when they compare two ArraySegment<T> objects. For two ArraySegment<T> objects to be considered equal, they must meet all of the following conditions:

- Reference the same array.
- Begin at the same index in the array.

- Have the same number of elements.

If you want to retrieve an element by its index in the ArraySegment<T> object, you must cast it to an IList<T> object and retrieve it or modify it by using the IList<T>.Item[] property. Note that this is not necessary in F#. The following example retrieves the element in an ArraySegment<T> object that delimits a section of a string array.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      String[] names = { "Adam", "Bruce", "Charles", "Daniel",
                         "Ebenezer", "Francis", "Gilbert",
                         "Henry", "Irving", "John", "Karl",
                         "Lucian", "Michael" };
      var partNames = new ArraySegment<string>(names, 2, 5);

      // Cast the ArraySegment object to an IList<string> and enumerate it.
      var list = (IList<string>) partNames;
      for (int ctr = 0; ctr <= list.Count - 1; ctr++)
         Console.WriteLine(list[ctr]);
   }
}
// The example displays the following output:
//    Charles
//    Daniel
//    Ebenezer
//    Francis
//    Gilbert
```

Constructors

Expand table

ArraySegment<T>(T[])	Initializes a new instance of the ArraySegment<T> structure that delimits all the elements in the specified array.
ArraySegment<T>(T[], Int32, Int32)	Initializes a new instance of the ArraySegment<T> structure that delimits the specified range of the elements in the specified array.

Properties

Expand table

Array	Gets the original array containing the range of elements that the array segment delimits.
Count	Gets the number of elements in the range delimited by the array segment.
Empty	Represents the empty array segment. This field is read-only.
Item[Int32]	Gets or sets the element at the specified index.
Offset	Gets the position of the first element in the range delimited by the array segment, relative to the start of the original array.

Methods

Expand table

CopyTo(ArraySegment<T>)	Copies the contents of this instance into the specified destination array segment of the same type `T`.
CopyTo(T[])	Copies the contents of this instance into the specified destination array of the same type `T`.
CopyTo(T[], Int32)	Copies the contents of this instance into the specified destination array of the same type `T`, starting at the specified destination index.
Equals(ArraySegment<T>)	Determines whether the specified ArraySegment<T> structure is equal to the current instance.
Equals(Object)	Determines whether the specified object is equal to the current instance.
GetEnumerator()	Returns an enumerator that can be used to iterate through the array segment.
GetHashCode()	Returns the hash code for the current instance.
Slice(Int32)	Forms a slice out of the current array segment starting at the specified index.
Slice(Int32, Int32)	Forms a slice of the specified length out of the current array segment starting at the specified index.
ToArray()	Copies the contents of this array segment into a new array.

Operators

Expand table

Equality(ArraySegment<T>, ArraySegment<T>)	Indicates whether two ArraySegment<T> structures are equal.
Implicit(T[] to ArraySegment<T>)	Defines an implicit conversion of an array of type `T` to an array segment of type `T`.
Inequality(ArraySegment<T>, ArraySegment<T>)	Indicates whether two ArraySegment<T> structures are unequal.

Explicit Interface Implementations

Expand table

ICollection<T>.Add(T)	Throws a NotSupportedException exception in all cases.
ICollection<T>.Clear()	Throws a NotSupportedException exception in all cases.
ICollection<T>.Contains(T)	Determines whether the array segment contains a specific value.
ICollection<T>.IsReadOnly	Gets a value that indicates whether the array segment is read-only.
ICollection<T>.Remove(T)	Throws a NotSupportedException exception in all cases.
IEnumerable.GetEnumerator()	Returns an enumerator that iterates through an array segment.
IEnumerable<T>.GetEnumerator()	Returns an enumerator that iterates through the array segment.
IList<T>.IndexOf(T)	Determines the index of a specific item in the array segment.
IList<T>.Insert(Int32, T)	Throws a NotSupportedException exception in all cases.
IList<T>.Item[Int32]	Gets or sets the element at the specified index.
IList<T>.RemoveAt(Int32)	Throws a NotSupportedException exception in all cases.
IReadOnlyList<T>.Item[Int32]	Gets the element at the specified index of the array segment.

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
AsReadOnly<T>(IList<T>)	Returns a read-only ReadOnlyCollection<T> wrapper for the specified list.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.

ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T>

	object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values

	that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.

Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a

	specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for

	each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default

	equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.

LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.

MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a

	sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.

OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key

	selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T>

	according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.

AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
AsMemory<T>(ArraySegment<T>)	Creates a new memory region over the portion of the target array segment.
AsMemory<T>(ArraySegment<T>, Int32)	Creates a new memory region over the portion of the target array segment starting at a specified position to the end of the segment.
AsMemory<T>(ArraySegment<T>, Int32, Int32)	Creates a new memory region over the portion of the target array segment beginning at a specified position with a specified length.
AsSpan<T>(ArraySegment<T>)	Creates a new span over a target array segment.
AsSpan<T>(ArraySegment<T>, Index)	Creates a new span over a portion of the target array segment beginning at a specified index and ending at the end of the segment.
AsSpan<T>(ArraySegment<T>, Int32)	Creates a new span over a portion of a target array segment from a specified position to the end of the segment.
AsSpan<T>(ArraySegment<T>, Int32, Int32)	Creates a new span over a portion of a target array segment from a specified position for a specified length.

AsSpan<T>(ArraySegment<T>, Range)	Creates a new span over a portion of a target array segment using the range start and end indexes.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.

InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Array

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArraySegment<T> structure.

Overloads

Expand table

ArraySegment<T>(T[])	Initializes a new instance of the ArraySegment<T> structure that delimits all the elements in the specified array.
ArraySegment<T>(T[], Int32, Int32)	Initializes a new instance of the ArraySegment<T> structure that delimits the specified range of the elements in the specified array.

Examples

The following code example passes an ArraySegment<T> structure to a method.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
      String[] myArr = { "The", "quick", "brown", "fox", "jumps", "over",
"the", "lazy", "dog" };

      // Display the initial contents of the array.
      Console.WriteLine( "The original array initially contains:" );
      PrintIndexAndValues( myArr );

      // Define an array segment that contains the entire array.
      ArraySegment<string> myArrSegAll = new ArraySegment<string>( myArr );

      // Display the contents of the ArraySegment.
```

```
      Console.WriteLine( "The first array segment (with all the array's
elements) contains:" );
      PrintIndexAndValues( myArrSegAll );

      // Define an array segment that contains the middle five values of the
array.
      ArraySegment<string> myArrSegMid = new ArraySegment<string>( myArr, 2,
5 );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The second array segment (with the middle five
elements) contains:" );
      PrintIndexAndValues( myArrSegMid );

      // Modify the fourth element of the first array segment myArrSegAll.
      myArrSegAll.Array[3] = "LION";

      // Display the contents of the second array segment myArrSegMid.
      // Note that the value of its second element also changed.
      Console.WriteLine( "After the first array segment is modified, the
second array segment now contains:" );
      PrintIndexAndValues( myArrSegMid );
   }

   public static void PrintIndexAndValues( ArraySegment<string> arrSeg )  {
      for ( int i = arrSeg.Offset; i < (arrSeg.Offset + arrSeg.Count); i++ )
{
         Console.WriteLine( "   [{0}] : {1}", i, arrSeg.Array[i] );
      }
      Console.WriteLine();
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The original array initially contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog
```

```
The first array segment (with all the array's elements) contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The second array segment (with the middle five elements) contains:
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the

After the first array segment is modified, the second array segment now
contains:
   [2] : brown
   [3] : LION
   [4] : jumps
   [5] : over
   [6] : the

*/
```

ArraySegment<T>(T[])

Initializes a new instance of the ArraySegment<T> structure that delimits all the elements in the specified array.

```
C#

public ArraySegment (T[] array);
```

Parameters

array T[]

The array to wrap.

Exceptions

ArgumentNullException

`array` is `null`.

Remarks

This constructor creates an ArraySegment<T> that delimits all the elements of `array`. That is, the Offset property of the ArraySegment<T> is 0 and its Count property is the length of `array`. To create an ArraySegment<T> that delimits only part of an array, use the ArraySegment<T>(T[], Int32, Int32) constructor.

The original array must be one-dimensional and must have zero-based indexing.

Multiple ArraySegment<T> instances can refer to the same original array and can overlap.

See also

- Array

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArraySegment<T>(T[], Int32, Int32)

Initializes a new instance of the ArraySegment<T> structure that delimits the specified range of the elements in the specified array.

C#

```
public ArraySegment (T[] array, int offset, int count);
```

Parameters

`array` T[]

The array containing the range of elements to delimit.

`offset` Int32

The zero-based index of the first element in the range.

`count` Int32

The number of elements in the range.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`offset` or `count` is less than 0.

ArgumentException

`offset` and `count` do not specify a valid range in `array`.

Remarks

The original array must be one-dimensional and must have zero-based indexing.

Multiple ArraySegment<T> instances can refer to the same original array and can overlap.

See also

- Array

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Array Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the original array containing the range of elements that the array segment delimits.

C#

```
public T[]? Array { get; }
```

Property Value

T[]

The original array that was passed to the constructor, and that contains the range delimited by the ArraySegment<T>.

Examples

The following code example passes an ArraySegment<T> to a method.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
      String[] myArr = { "The", "quick", "brown", "fox", "jumps", "over",
"the", "lazy", "dog" };

      // Display the initial contents of the array.
      Console.WriteLine( "The original array initially contains:" );
      PrintIndexAndValues( myArr );

      // Define an array segment that contains the entire array.
      ArraySegment<string> myArrSegAll = new ArraySegment<string>( myArr );
```

```
      // Display the contents of the ArraySegment.
      Console.WriteLine( "The first array segment (with all the array's
elements) contains:" );
      PrintIndexAndValues( myArrSegAll );

      // Define an array segment that contains the middle five values of the
array.
      ArraySegment<string> myArrSegMid = new ArraySegment<string>( myArr, 2,
5 );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The second array segment (with the middle five
elements) contains:" );
      PrintIndexAndValues( myArrSegMid );

      // Modify the fourth element of the first array segment myArrSegAll.
      myArrSegAll.Array[3] = "LION";

      // Display the contents of the second array segment myArrSegMid.
      // Note that the value of its second element also changed.
      Console.WriteLine( "After the first array segment is modified, the
second array segment now contains:" );
      PrintIndexAndValues( myArrSegMid );
   }

   public static void PrintIndexAndValues( ArraySegment<string> arrSeg )  {
      for ( int i = arrSeg.Offset; i < (arrSeg.Offset + arrSeg.Count); i++ )
{
         Console.WriteLine( "   [{0}] : {1}", i, arrSeg.Array[i] );
      }
      Console.WriteLine();
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The original array initially contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog
```

```
The first array segment (with all the array's elements) contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The second array segment (with the middle five elements) contains:
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the

After the first array segment is modified, the second array segment now
contains:
   [2] : brown
   [3] : LION
   [4] : jumps
   [5] : over
   [6] : the

*/
```

Remarks

The Array property returns the original array, not a copy of the array; therefore, changes made through the property are made directly to the original array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Array

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Count Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the range delimited by the array segment.

C#

```
public int Count { get; }
```

Property Value

Int32

The number of elements in the range delimited by the ArraySegment<T>.

Implements

Count , Count

Examples

The following code example passes an ArraySegment<T> structure to a method.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
      String[] myArr = { "The", "quick", "brown", "fox", "jumps", "over",
"the", "lazy", "dog" };

      // Display the initial contents of the array.
      Console.WriteLine( "The original array initially contains:" );
      PrintIndexAndValues( myArr );
```

```
      // Define an array segment that contains the entire array.
      ArraySegment<string> myArrSegAll = new ArraySegment<string>( myArr );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The first array segment (with all the array's
elements) contains:" );
      PrintIndexAndValues( myArrSegAll );

      // Define an array segment that contains the middle five values of the
array.
      ArraySegment<string> myArrSegMid = new ArraySegment<string>( myArr, 2,
5 );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The second array segment (with the middle five
elements) contains:" );
      PrintIndexAndValues( myArrSegMid );

      // Modify the fourth element of the first array segment myArrSegAll.
      myArrSegAll.Array[3] = "LION";

      // Display the contents of the second array segment myArrSegMid.
      // Note that the value of its second element also changed.
      Console.WriteLine( "After the first array segment is modified, the
second array segment now contains:" );
      PrintIndexAndValues( myArrSegMid );
   }

   public static void PrintIndexAndValues( ArraySegment<string> arrSeg )  {
      for ( int i = arrSeg.Offset; i < (arrSeg.Offset + arrSeg.Count); i++ )
{
         Console.WriteLine( "   [{0}] : {1}", i, arrSeg.Array[i] );
      }
      Console.WriteLine();
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The original array initially contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
```

```
   [6] : the
   [7] : lazy
   [8] : dog

The first array segment (with all the array's elements) contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The second array segment (with the middle five elements) contains:
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the

After the first array segment is modified, the second array segment now
contains:
   [2] : brown
   [3] : LION
   [4] : jumps
   [5] : over
   [6] : the

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Array
- Offset

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.Empty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the empty array segment. This field is read-only.

C#

```
public static ArraySegment<T> Empty { get; }
```

Property Value

ArraySegment<T>

An array segment of type `T` containing zero elements.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the element at the specified index.

```
C#
public T this[int index] { get; set; }
```

Parameters

`index` Int32

The zero-based index of the element to get or set.

Property Value

T

The element at the specified index.

Implements

Item[Int32]

Exceptions

ArgumentOutOfRangeException

`index` is not a valid index in the ArraySegment<T>.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Offset Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the position of the first element in the range delimited by the array segment, relative to the start of the original array.

C#

```
public int Offset { get; }
```

Property Value

Int32

The position of the first element in the range delimited by the ArraySegment<T>, relative to the start of the original array.

Examples

The following code example passes an ArraySegment<T> structure to a method.

C#

```
using System;

public class SamplesArray  {

   public static void Main()  {

      // Create and initialize a new string array.
      String[] myArr = { "The", "quick", "brown", "fox", "jumps", "over",
"the", "lazy", "dog" };

      // Display the initial contents of the array.
      Console.WriteLine( "The original array initially contains:" );
      PrintIndexAndValues( myArr );

      // Define an array segment that contains the entire array.
      ArraySegment<string> myArrSegAll = new ArraySegment<string>( myArr );
```

```
      // Display the contents of the ArraySegment.
      Console.WriteLine( "The first array segment (with all the array's
elements) contains:" );
      PrintIndexAndValues( myArrSegAll );

      // Define an array segment that contains the middle five values of the
array.
      ArraySegment<string> myArrSegMid = new ArraySegment<string>( myArr, 2,
5 );

      // Display the contents of the ArraySegment.
      Console.WriteLine( "The second array segment (with the middle five
elements) contains:" );
      PrintIndexAndValues( myArrSegMid );

      // Modify the fourth element of the first array segment myArrSegAll.
      myArrSegAll.Array[3] = "LION";

      // Display the contents of the second array segment myArrSegMid.
      // Note that the value of its second element also changed.
      Console.WriteLine( "After the first array segment is modified, the
second array segment now contains:" );
      PrintIndexAndValues( myArrSegMid );
   }

   public static void PrintIndexAndValues( ArraySegment<string> arrSeg )  {
      for ( int i = arrSeg.Offset; i < (arrSeg.Offset + arrSeg.Count); i++ )
{
         Console.WriteLine( "   [{0}] : {1}", i, arrSeg.Array[i] );
      }
      Console.WriteLine();
   }

   public static void PrintIndexAndValues( String[] myArr )  {
      for ( int i = 0; i < myArr.Length; i++ )  {
         Console.WriteLine( "   [{0}] : {1}", i, myArr[i] );
      }
      Console.WriteLine();
   }
}


/*
This code produces the following output.

The original array initially contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
```

```
   [8] : dog

The first array segment (with all the array's elements) contains:
   [0] : The
   [1] : quick
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the
   [7] : lazy
   [8] : dog

The second array segment (with the middle five elements) contains:
   [2] : brown
   [3] : fox
   [4] : jumps
   [5] : over
   [6] : the

After the first array segment is modified, the second array segment now
contains:
   [2] : brown
   [3] : LION
   [4] : jumps
   [5] : over
   [6] : the

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Array
- Count

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.CopyTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CopyTo(ArraySegment<T>)	Copies the contents of this instance into the specified destination array segment of the same type `T`.
CopyTo(T[])	Copies the contents of this instance into the specified destination array of the same type `T`.
CopyTo(T[], Int32)	Copies the contents of this instance into the specified destination array of the same type `T`, starting at the specified destination index.

CopyTo(ArraySegment<T>)

Copies the contents of this instance into the specified destination array segment of the same type `T`.

C#

```
public void CopyTo (ArraySegment<T> destination);
```

Parameters

`destination` ArraySegment<T>

The array segment into which the contents of this instance will be copied.

Exceptions

InvalidOperationException

The underlying array of this instance is `null`.

-or-

The underlying array of `destination` is `null`.

ArgumentException

The length of the underlying array of this instance is larger than the length of the underlying array of `destination`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

CopyTo(T[])

Copies the contents of this instance into the specified destination array of the same type `T`.

C#

```
public void CopyTo (T[] destination);
```

Parameters

`destination` T[]

The array of type `T` into which the contents of this instance will be copied.

Exceptions

InvalidOperationException

The underlying array of this instance is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

CopyTo(T[], Int32)

Copies the contents of this instance into the specified destination array of the same type `T`, starting at the specified destination index.

C#

```
public void CopyTo (T[] destination, int destinationIndex);
```

Parameters

`destination` T[]

The array of type `T` into which the contents of this instance will be copied.

`destinationIndex` Int32

The index in `destination` at which storing begins.

Implements

CopyTo(T[], Int32)

Exceptions

InvalidOperationException

The underlying array of this instance is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two objects are equal.

Overloads

Expand table

Equals(ArraySegment<T>)	Determines whether the specified ArraySegment<T> structure is equal to the current instance.
Equals(Object)	Determines whether the specified object is equal to the current instance.

Equals(ArraySegment<T>)

Determines whether the specified ArraySegment<T> structure is equal to the current instance.

C#

```
public bool Equals (ArraySegment<T> obj);
```

Parameters

`obj` ArraySegment<T>

The structure to compare with the current instance.

Returns

Boolean

`true` if the specified ArraySegment<T> structure is equal to the current instance; otherwise, `false`.

Remarks

Two ArraySegment<T> objects are considered to be equal if all the following conditions are met:

- They reference the same array.
- They begin at the same index in the array.
- They have the same number of elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Determines whether the specified object is equal to the current instance.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to be compared with the current instance.

Returns

Boolean

`true` if the specified object is a ArraySegment<T> structure and is equal to the current instance; otherwise, `false`.

Remarks

Two ArraySegment<T> objects are considered to be equal if all the following conditions are met:

- They reference the same array.
- They begin at the same index in the array.
- They have the same number of elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator that can be used to iterate through the array segment.

C#

```
public ArraySegment<T>.Enumerator GetEnumerator ();
```

Returns

ArraySegment<T>.Enumerator

An enumerator that can be used to iterate through the array segment.

Exceptions

InvalidOperationException

The underlying array is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ArraySegment<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ArraySegment<T>.Slice Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Slice(Int32)	Forms a slice out of the current array segment starting at the specified index.
Slice(Int32, Int32)	Forms a slice of the specified length out of the current array segment starting at the specified index.

Slice(Int32)

Forms a slice out of the current array segment starting at the specified index.

C#

```
public ArraySegment<T> Slice (int index);
```

Parameters

`index` Int32

The index at which to begin the slice.

Returns

ArraySegment<T>

An array segment that consists of all elements of the current array segment from `index` to the end of the array segment.

Exceptions

InvalidOperationException

The underlying array of this instance is `null`.

ArgumentOutOfRangeException

`index` is greater than the length of the underlying array of this instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Slice(Int32, Int32)

Forms a slice of the specified length out of the current array segment starting at the specified index.

C#

```
public ArraySegment<T> Slice (int index, int count);
```

Parameters

`index` Int32

The index at which to begin the slice.

`count` Int32

The desired length of the slice.

Returns

ArraySegment<T>

An array segment of `count` elements starting at `index`.

Exceptions

InvalidOperationException

The underlying array of this instance is `null`.

ArgumentOutOfRangeException

`index` is greater than the length of the underlying array of this instance.

-or-

`count` is greater than the length of the underlying array of this instance - `index`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.ToArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this array segment into a new array.

C#

```
public T[] ToArray ();
```

Returns

T[]

An array containing the data in the current array segment.

Exceptions

InvalidOperationException

`default(ArraySegment<T>)` cannot be converted to an array.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ArraySegment<T>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two ArraySegment<T> structures are equal.

```
C#
public static bool operator == (ArraySegment<T> a, ArraySegment<T> b);
```

Parameters

a ArraySegment<T>

The structure on the left side of the equality operator.

b ArraySegment<T>

The structure on the right side of the equality operator.

Returns

Boolean

`true` if `a` is equal to `b`; otherwise, `false`.

Remarks

Two ArraySegment<T> objects are considered to be equal if all the following conditions are met:

- They reference the same array.
- They begin at the same index.
- They have the same number of elements.

The equivalent method for this operator is ArraySegment<T>.Equals(ArraySegment<T>)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.Implicit(T[] to ArraySegment<T>) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an implicit conversion of an array of type `T` to an array segment of type `T`.

```
C#
public static implicit operator ArraySegment<T> (T[] array);
```

Parameters

`array` T[]

The array to convert.

Returns

ArraySegment<T>

An array segment representation of the array.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ArraySegment<T>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two ArraySegment<T> structures are unequal.

```
C#
public static bool operator != (ArraySegment<T> a, ArraySegment<T> b);
```

Parameters

a ArraySegment<T>

The structure on the left side of the inequality operator.

b ArraySegment<T>

The structure on the right side of the inequality operator.

Returns

Boolean

`true` if `a` is not equal to `b`; otherwise, `false`.

Remarks

Two ArraySegment<T> objects are considered to be unequal if they are of the same closed generic type but any of the following conditions is true:

- They do not reference the same array.
- They have different starting indexes.
- They have a different number of elements.

The equivalent method for this operator is ArraySegment<T>.Equals(ArraySegment<T>)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array Segment<T>.ICollection<T>.Add(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws a NotSupportedException exception in all cases.

```
C#
void ICollection<T>.Add (T item);
```

Parameters

`item` T

The object to add to the array segment.

Implements

Add(T)

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws a NotSupportedException because an array segment cannot be expanded.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.ICollection<T>.Clear Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws a NotSupportedException exception in all cases.

```
C#
void ICollection<T>.Clear ();
```

Implements

Clear()

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws a NotSupportedException because an array segment cannot be contracted.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Array Segment<T>.ICollection<T>.Contains(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the array segment contains a specific value.

```
C#
bool ICollection<T>.Contains (T item);
```

Parameters

`item` T

The object to locate in the array segment.

Returns

Boolean

`true` if `item` is found in the array segment; otherwise, `false`.

Implements

Contains(T)

Exceptions

InvalidOperationException

The underlying array is `null`.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an ICollection<T> interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.ICollection<T>.IsReadOnly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the array segment is read-only.

C#

```
bool System.Collections.Generic.ICollection<T>.IsReadOnly { get; }
```

Property Value

Boolean

`true` if the array segment is read-only; otherwise, `false`.

Implements

IsReadOnly

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an ICollection<T> interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Array Segment<T>.ICollection<T>.Remove(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws a NotSupportedException exception in all cases.

```
C#
bool ICollection<T>.Remove (T item);
```

Parameters

`item` T

The object to remove from the array segment.

Returns

Boolean

Always throws an exception.

Implements

Remove(T)

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws a NotSupportedException because an array segment cannot be contracted.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.IEnumerable<T>.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator that iterates through the array segment.

```
C#
System.Collections.Generic.IEnumerator<T> IEnumerable<T>.GetEnumerator ();
```

Returns

IEnumerator<T>

An enumerator that can be used to iterate through the array segment.

Implements

GetEnumerator()

Exceptions

InvalidOperationException

The underlying array is `null`.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an IEnumerable<T> interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.IList<T>.IndexOf(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines the index of a specific item in the array segment.

```
C#
int IList<T>.IndexOf (T item);
```

Parameters

`item` T

The object to locate in the array segment.

Returns

Int32

The index of `item` if found in the list; otherwise, -1.

Implements

IndexOf(T)

Exceptions

InvalidOperationException

The underlying array is `null`.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an IList<T> interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.IList<T>.Insert(Int32, T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws a NotSupportedException exception in all cases.

```
C#
void IList<T>.Insert (int index, T item);
```

Parameters

index Int32

The zero-based index at which `item` should be inserted.

item T

The object to insert into the array segment.

Implements

Insert(Int32, T)

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws a NotSupportedException because an array segment cannot be expanded.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.IList<T>.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the element at the specified index.

```
C#

T System.Collections.Generic.IList<T>.Item[int index] { get; set; }
```

Parameters

`index` Int32

The zero-based index of the element to get or set.

Property Value

T

The element at the specified index.

Implements

Item[Int32]

Exceptions

ArgumentOutOfRangeException

`index` is not a valid index in the ArraySegment<T>.

NotSupportedException

The property is set and the array segment is read-only.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an IList<T> interface, as the following example shows.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      String[] names = { "Adam", "Bruce", "Charles", "Daniel",
                         "Ebenezer", "Francis", "Gilbert",
                         "Henry", "Irving", "John", "Karl",
                         "Lucian", "Michael" };
      var partNames = new ArraySegment<string>(names, 2, 5);

      // Cast the ArraySegment object to an IList<string> and enumerate it.
      var list = (IList<string>) partNames;
      for (int ctr = 0; ctr <= list.Count - 1; ctr++)
         Console.WriteLine(list[ctr]);
   }
}
// The example displays the following output:
//    Charles
//    Daniel
//    Ebenezer
//    Francis
//    Gilbert
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArraySegment<T>.IList<T>.RemoveAt(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Throws a NotSupportedException exception in all cases.

```
C#
void IList<T>.RemoveAt (int index);
```

Parameters

`index` Int32

The zero-based index of the item to remove.

Implements

RemoveAt(Int32)

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws a NotSupportedException because an array segment cannot be contracted.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.IReadOnlyList<T>.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the element at the specified index of the array segment.

```
C#

T System.Collections.Generic.IReadOnlyList<T>.Item[int index] { get; }
```

Parameters

index Int32

The zero-based index of the element to get.

Property Value

T

The element at the specified index.

Implements

Item[Int32]

Exceptions

ArgumentOutOfRangeException

`index` is not a valid index in the ArraySegment<T>.

InvalidOperationException

The underlying array is `null`.

NotSupportedException

The property is set.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an IReadOnlyList<T> interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArraySegment<T>.IEnumerable.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator that iterates through an array segment.

```
C#
System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An enumerator that can be used to iterate through the array segment.

Implements

GetEnumerator()

Exceptions

InvalidOperationException

The underlying array is `null`.

Remarks

This member is an explicit interface member implementation. It can be used only when the ArraySegment<T> instance is cast to an IEnumerable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ArrayTypeMismatchException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt is made to store an element of the wrong type within an array.

C#

```
public class ArrayTypeMismatchException : SystemException
```

Inheritance Object → Exception → SystemException → ArrayTypeMismatchException

Examples

The following code example demonstrates scenarios where ArrayTypeMismatchException is thrown.

C#

```
using System;

namespace ArrayTypeMismatch
{
    class Class1
    {
        static void Main(string[] args)
        {
            string[] names = {"Dog", "Cat", "Fish"};
            Object[] objs  = (Object[]) names;

            try
            {
                objs[2] = "Mouse";

                foreach (object animalName in objs)
                {
                    System.Console.WriteLine(animalName);
                }
            }
            catch (System.ArrayTypeMismatchException)
```

```
            {
                // Not reached; "Mouse" is of the correct type.
                System.Console.WriteLine("Exception Thrown.");
            }

            try
            {
                Object obj = (Object) 13;
                objs[2] = obj;
            }
            catch (System.ArrayTypeMismatchException)
            {
                // Always reached, 13 is not a string.
                System.Console.WriteLine(
                    "New element is not of the correct type.");
            }

            // Set objs to an array of objects instead of
            // an array of strings.
            objs  = new Object[3];
            try
            {
                objs[0] = (Object) "Turtle";
                objs[1] = (Object) 12;
                objs[2] = (Object) 2.341;

                foreach (object element in objs)
                {
                    System.Console.WriteLine(element);
                }
            }
            catch (System.ArrayTypeMismatchException)
            {
                // ArrayTypeMismatchException is not thrown this time.
                System.Console.WriteLine("Exception Thrown.");
            }
        }
    }
}
```

Remarks

ArrayTypeMismatchException is thrown when the system cannot convert the element to the type declared for the array. For example, an element of type String cannot be stored in an Int32 array because conversion between these types is not supported. It is generally unnecessary for applications to throw this exception.

The following Microsoft intermediate language (MSIL) instructions throw ArrayTypeMismatchException:

- `ldelem.<type>`
- `ldelema`
- `stelem.<type>`

ArrayTypeMismatchException uses the HRESULT COR_E_ARRAYTYPEMISMATCH, which has the value 0x80131503.

For a list of initial property values for an instance of ArrayTypeMismatchException, see the ArrayTypeMismatchException constructors.

Constructors

Expand table

ArrayTypeMismatchException()	Initializes a new instance of the ArrayTypeMismatchException class.
ArrayTypeMismatchException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArrayTypeMismatchException class with serialized data.
ArrayTypeMismatchException(String)	Initializes a new instance of the ArrayTypeMismatchException class with a specified error message.
ArrayTypeMismatchException(String, Exception)	Initializes a new instance of the ArrayTypeMismatchException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

	(Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ArrayTypeMismatchException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ArrayTypeMismatchException class.

Overloads

Expand table

ArrayTypeMismatchException()	Initializes a new instance of the ArrayTypeMismatchException class.
ArrayTypeMismatchException(String)	Initializes a new instance of the ArrayTypeMismatchException class with a specified error message.
ArrayTypeMismatchException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ArrayTypeMismatchException class with serialized data.
ArrayTypeMismatchException(String, Exception)	Initializes a new instance of the ArrayTypeMismatchException class with a specified error message and a reference to the inner exception that is the cause of this exception.

ArrayTypeMismatchException()

Initializes a new instance of the ArrayTypeMismatchException class.

C#

```
public ArrayTypeMismatchException ();
```

Examples

The following example demonstrates the ArrayTypeMismatchException() constructor of the ArrayTypeMismatchException class. It contains a function which takes two arrays as arguments and checks whether the two arrays are of the same type. If the arrays are of different types, a new ArrayTypeMismatchException is thrown and then caught in the calling method.

C#

```
using System;

public class ArrayTypeMisMatchConst
{
   public void CopyArray(Array myArray,Array myArray1)
   {
      string typeArray1 = myArray.GetType().ToString();
      string typeArray2 = myArray1.GetType().ToString();
      // Check whether the two arrays are of same type or not.
      if(typeArray1==typeArray2)
      {
         // Copy the values from one array to another.
         myArray.SetValue("Name: "+myArray1.GetValue(0),0);
         myArray.SetValue("Name: "+myArray1.GetValue(1),1);
      }
      else
      {
         // Throw an exception of type 'ArrayTypeMismatchException'.
         throw new ArrayTypeMismatchException();
      }
   }
   static void Main()
   {
      try
      {
         string[] myStringArray = new string[2];
         myStringArray.SetValue("Jones",0);
         myStringArray.SetValue("John",1);

         int[] myIntArray = new int[2];
         ArrayTypeMisMatchConst myArrayType = new
ArrayTypeMisMatchConst();
         myArrayType.CopyArray(myStringArray,myIntArray);
      }
      catch(ArrayTypeMismatchException e)
      {
         Console.WriteLine("The Exception is :"+e);
      }
   }
}
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Source array type cannot be assigned to destination array type." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ArrayTypeMismatchException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArrayTypeMismatchException(String)

Initializes a new instance of the ArrayTypeMismatchException class with a specified error message.

C#

```
public ArrayTypeMismatchException (string? message);
```

Parameters

`message` String

A String that describes the error.

Examples

The following example demonstrates the ArrayTypeMismatchException(String) constructor of the ArrayTypeMismatchException class. It contains a function which takes two arrays as arguments and checks whether the two arrays are of the same type. If the arrays are of different types, a new ArrayTypeMismatchException is thrown and then caught in the calling method.

C#

```
using System;

public class ArrayTypeMisMatchConst
{
   public void CopyArray(Array myArray,Array myArray1)
   {
      string typeArray1 = myArray.GetType().ToString();
      string typeArray2 = myArray1.GetType().ToString();
      // Check whether the two arrays are of same type or not.
      if(typeArray1==typeArray2)
      {
         // Copies the values from one array to another.
         myArray.SetValue("Name: "+myArray1.GetValue(0),0);
         myArray.SetValue("Name: "+myArray1.GetValue(1),1);
      }
      else
      {
          // Throw an exception of type 'ArrayTypeMismatchException' with
a message string as parameter.
         throw new ArrayTypeMismatchException("The Source and destination
arrays are not of same type.");
      }
   }

   static void Main()
   {
      try
      {
         string[] myStringArray = new string[2];
         myStringArray.SetValue("Jones",0);
         myStringArray.SetValue("John",1);
         int[] myIntArray = new int[2];
         ArrayTypeMisMatchConst myArrayType = new
ArrayTypeMisMatchConst();
         myArrayType.CopyArray(myStringArray,myIntArray);
```

```
        }
        catch(ArrayTypeMismatchException e)
        {
           Console.WriteLine("The Exception Message is : " + e.Message);
        }
    }
}
```

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of ArrayTypeMismatchException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ArrayTypeMismatchException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ArrayTypeMismatchException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ArrayTypeMismatchException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ArrayTypeMismatchException(String, Exception)

Initializes a new instance of the ArrayTypeMismatchException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public ArrayTypeMismatchException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Examples

The following code example demonstrates the ArrayTypeMismatchException constructor of the ArrayTypeMismatchException class. It contains a function that takes two arrays as arguments and checks whether the two arrays are of the same type. If the arrays are of different types, a new ArrayTypeMismatchException is thrown and then caught in the calling method.

C#

```
using System;

public class ArrayTypeMisMatchConst
{
   public void CopyArray(Array myArray,Array myArray1)
   {
      try
      {
         // Copies the value of one array into another array.
         myArray.SetValue(myArray1.GetValue(0),0);
         myArray.SetValue(myArray1.GetValue(1),1);
      }
      catch(Exception e)
      {
         // Throw an exception of with a message and innerexception.
         throw new ArrayTypeMismatchException("The Source and destination
arrays are of not same type.", e);
      }
   }
   static void Main()
   {
         try
         {
            string[] myStringArray = new string[2];
            myStringArray.SetValue("Jones",0);
            myStringArray.SetValue("John",1);
            int[] myIntArray = new int[2];
            ArrayTypeMisMatchConst myArrayType = new
ArrayTypeMisMatchConst();
            myArrayType.CopyArray(myStringArray,myIntArray);
         }
         catch(ArrayTypeMismatchException e)
         {
            Console.WriteLine("The Exception Message is : "+e.Message);
            Console.WriteLine("The Inner exception is
:"+e.InnerException);
         }
      }
}
```

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ArrayTypeMismatchException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AssemblyLoadEventArgs Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides data for the AssemblyLoad event.

C#

```
public class AssemblyLoadEventArgs : EventArgs
```

Inheritance Object → EventArgs → AssemblyLoadEventArgs

Constructors

Expand table

AssemblyLoadEventArgs(Assembly)	Initializes a new instance of the AssemblyLoadEventArgs class using the specified Assembly.

Properties

Expand table

LoadedAssembly	Gets an Assembly that represents the currently loaded assembly.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)

GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AssemblyLoadEventArgs(Assembly) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the AssemblyLoadEventArgs class using the specified Assembly.

C#

```
public AssemblyLoadEventArgs (System.Reflection.Assembly loadedAssembly);
```

Parameters

`loadedAssembly` Assembly

An instance that represents the currently loaded assembly.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

AssemblyLoadEventArgs.LoadedAssembly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an Assembly that represents the currently loaded assembly.

C#

```
public System.Reflection.Assembly LoadedAssembly { get; }
```

Property Value

Assembly

An instance of Assembly that represents the currently loaded assembly.

Remarks

The value of this property is set by the common language runtime when the AppDomain.AssemblyLoad event occurs.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

AssemblyLoadEventHandler Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that handles the AssemblyLoad event of an AppDomain.

C#

```
public delegate void AssemblyLoadEventHandler(object? sender,
AssemblyLoadEventArgs args);
```

Parameters

sender Object

The source of the event.

args AssemblyLoadEventArgs

An AssemblyLoadEventArgs that contains the event data.

Remarks

An AssemblyLoadEventHandler is used to specify the methods that are invoked in response to an AssemblyLoad event. To associate an instance of AssemblyLoadEventHandler with an event, add the instance to the event. The methods referenced by the AssemblyLoadEventHandler are invoked whenever an assembly is loaded, until the AssemblyLoadEventHandler is removed from the event.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AsyncCallback Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

References a method to be called when a corresponding asynchronous operation completes.

C#

```
public delegate void AsyncCallback(IAsyncResult ar);
```

Parameters

ar IAsyncResult

The result of the asynchronous operation.

Examples

The following code example demonstrates using asynchronous methods in the Dns class to retrieve Domain Name System (DNS) information for user-specified computers. This example creates an AsyncCallback delegate that references the `ProcessDnsInformation` method. This method is called once for each asynchronous request for DNS information.

C#

```
/*
The following example demonstrates using asynchronous methods to
get Domain Name System information for the specified host computers.
This example uses a delegate to obtain the results of each asynchronous
operation.
*/

using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections.Specialized;
using System.Collections;
```

```
namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class UseDelegateForAsyncCallback
    {
        static int requestCounter;
        static ArrayList hostData = new ArrayList();
        static StringCollection hostNames = new StringCollection();
        static void UpdateUserInterface()
        {
            // Print a message to indicate that the application
            // is still working on the remaining requests.
            Console.WriteLine("{0} requests remaining.", requestCounter);
        }
        public static void Main()
        {
            // Create the delegate that will process the results of the
            // asynchronous request.
            AsyncCallback callBack = new
AsyncCallback(ProcessDnsInformation);
            string host;
            do
            {
                Console.Write(" Enter the name of a host computer or <enter>
to finish: ");
                host = Console.ReadLine();
                if (host.Length > 0)
                {
                    // Increment the request counter in a thread safe
manner.
                    Interlocked.Increment(ref requestCounter);
                    // Start the asynchronous request for DNS information.
                    Dns.BeginGetHostEntry(host, callBack, host);
                 }
            } while (host.Length > 0);
            // The user has entered all of the host names for lookup.
            // Now wait until the threads complete.
            while (requestCounter > 0)
            {
                UpdateUserInterface();
            }
            // Display the results.
            for (int i = 0; i< hostNames.Count; i++)
            {
                object data = hostData [i];
                string message = data as string;
                // A SocketException was thrown.
                if (message != null)
                {
                    Console.WriteLine("Request for {0} returned message:
{1}",
                        hostNames[i], message);
                    continue;
                }
                // Get the results.
                IPHostEntry h = (IPHostEntry) data;
```

```
                string[] aliases = h.Aliases;
                IPAddress[] addresses = h.AddressList;
                if (aliases.Length > 0)
                {
                    Console.WriteLine("Aliases for {0}", hostNames[i]);
                    for (int j = 0; j < aliases.Length; j++)
                    {
                        Console.WriteLine("{0}", aliases[j]);
                    }
                }
                if (addresses.Length > 0)
                {
                    Console.WriteLine("Addresses for {0}", hostNames[i]);
                    for (int k = 0; k < addresses.Length; k++)
                    {
                        Console.WriteLine("{0}",addresses[k].ToString());
                    }
                }
            }
        }

        // The following method is called when each asynchronous operation
completes.
        static void ProcessDnsInformation(IAsyncResult result)
        {
            string hostName = (string) result.AsyncState;
            hostNames.Add(hostName);
            try
            {
                // Get the results.
                IPHostEntry host = Dns.EndGetHostEntry(result);
                hostData.Add(host);
            }
            // Store the exception message.
            catch (SocketException e)
            {
                hostData.Add(e.Message);
            }
            finally
            {
                // Decrement the request counter in a thread-safe manner.
                Interlocked.Decrement(ref requestCounter);
            }
        }
    }
}
```

Remarks

Use an AsyncCallback delegate to process the results of an asynchronous operation in a separate thread. The AsyncCallback delegate represents a callback method that is called

when the asynchronous operation completes. The callback method takes an IAsyncResult parameter, which is subsequently used to obtain the results of the asynchronous operation.

For more information about asynchronous programming, see Using an AsyncCallback Delegate to End an Asynchronous Operation and Using an AsyncCallback Delegate and State Object in Event-based Asynchronous Pattern (EAP).

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IAsyncResult
- Using an AsyncCallback Delegate to End an Asynchronous Operation
- Using an AsyncCallback Delegate and State Object
- Calling Synchronous Methods Asynchronously
- Event-based Asynchronous Pattern (EAP)
- Handling and Raising Events

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Attribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the base class for custom attributes.

C#

```
[System.AttributeUsage(System.AttributeTargets.All, AllowMultiple=false,
Inherited=true)]
public abstract class Attribute
```

Inheritance Object → Attribute

Derived Microsoft.Extensions.Compliance.Classification.DataClassificationAttribute
Microsoft.Extensions.Configuration.ConfigurationKeyNameAttribute
Microsoft.Extensions.Configuration.UserSecrets.UserSecretsIdAttribute
Microsoft.Extensions.DependencyInjection.ActivatorUtilitiesConstructorAttribute
Microsoft.Extensions.DependencyInjection.FromKeyedServicesAttribute
More...

Attributes AttributeUsageAttribute

Examples

The following code example demonstrates the usage of Attribute.

C#

```
using System;
using System.Reflection;

// An enumeration of animals. Start at 1 (0 = uninitialized).
public enum Animal {
    // Pets.
    Dog = 1,
    Cat,
```

```
    Bird,
}

// A custom attribute to allow a target to have a pet.
public class AnimalTypeAttribute : Attribute {
    // The constructor is called when the attribute is set.
    public AnimalTypeAttribute(Animal pet) {
        thePet = pet;
    }

    // Keep a variable internally ...
    protected Animal thePet;

    // .. and show a copy to the outside world.
    public Animal Pet {
        get { return thePet; }
        set { thePet = value; }
    }
}

// A test class where each method has its own pet.
class AnimalTypeTestClass {
    [AnimalType(Animal.Dog)]
    public void DogMethod() {}

    [AnimalType(Animal.Cat)]
    public void CatMethod() {}

    [AnimalType(Animal.Bird)]
    public void BirdMethod() {}
}

class DemoClass {
    static void Main(string[] args) {
        AnimalTypeTestClass testClass = new AnimalTypeTestClass();
        Type type = testClass.GetType();
        // Iterate through all the methods of the class.
        foreach(MethodInfo mInfo in type.GetMethods()) {
            // Iterate through all the Attributes for each method.
            foreach (Attribute attr in
                Attribute.GetCustomAttributes(mInfo)) {
                // Check for the AnimalType attribute.
                if (attr.GetType() == typeof(AnimalTypeAttribute))
                    Console.WriteLine(
                        "Method {0} has a pet {1} attribute.",
                        mInfo.Name, ((AnimalTypeAttribute)attr).Pet);
            }
        }
    }
}
/*
 * Output:
 * Method DogMethod has a pet Dog attribute.
 * Method CatMethod has a pet Cat attribute.
```

```
 * Method BirdMethod has a pet Bird attribute.
 */
```

Remarks

The Attribute class associates predefined system information or user-defined custom information with a target element. A target element can be an assembly, class, constructor, delegate, enum, event, field, interface, method, portable executable file module, parameter, property, return value, struct, or another attribute.

Information provided by an attribute is also known as metadata. Metadata can be examined at run time by your application to control how your program processes data, or before run time by external tools to control how your application itself is processed or maintained. For example, .NET predefines and uses attribute types to control run-time behavior, and some programming languages use attribute types to represent language features not directly supported by the .NET common type system.

All attribute types derive directly or indirectly from the Attribute class. Attributes can be applied to any target element; multiple attributes can be applied to the same target element; and attributes can be inherited by an element derived from a target element. Use the AttributeTargets class to specify the target element to which the attribute is applied.

The Attribute class provides convenient methods to retrieve and test custom attributes. For more information about using attributes, see Applying Attributes and Attributes.

Constructors

Expand table

Attribute()	Initializes a new instance of the Attribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.
GetCustomAttribute(Assembly, Type)	Retrieves a custom attribute applied to a specified assembly. Parameters specify the assembly and the type of the custom attribute to search for.
GetCustomAttribute(Assembly, Type, Boolean)	Retrieves a custom attribute applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttribute(MemberInfo, Type)	Retrieves a custom attribute applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
GetCustomAttribute(MemberInfo, Type, Boolean)	Retrieves a custom attribute applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttribute(Module, Type)	Retrieves a custom attribute applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
GetCustomAttribute(Module, Type, Boolean)	Retrieves a custom attribute applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttribute(ParameterInfo, Type)	Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.
GetCustomAttribute(ParameterInfo, Type, Boolean)	Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
GetCustomAttributes(Assembly)	Retrieves an array of the custom attributes applied to an assembly. A parameter specifies the assembly.
GetCustomAttributes(Assembly, Boolean)	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and an ignored search option.
GetCustomAttributes(Assembly, Type)	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.
GetCustomAttributes(Assembly, Type,	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, the type of the

Boolean)	custom attribute to search for, and an ignored search option.
GetCustomAttributes(MemberInfo)	Retrieves an array of the custom attributes applied to a member of a type. A parameter specifies the member.
GetCustomAttributes(MemberInfo, Boolean)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttributes(MemberInfo, Type)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
GetCustomAttributes(MemberInfo, Type, Boolean)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttributes(Module)	Retrieves an array of the custom attributes applied to a module. A parameter specifies the module.
GetCustomAttributes(Module, Boolean)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and an ignored search option.
GetCustomAttributes(Module, Type)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
GetCustomAttributes(Module, Type, Boolean)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttributes(ParameterInfo)	Retrieves an array of the custom attributes applied to a method parameter. A parameter specifies the method parameter.
GetCustomAttributes(ParameterInfo, Boolean)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and whether to search ancestors of the method parameter.
GetCustomAttributes(ParameterInfo, Type)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.
GetCustomAttributes(ParameterInfo, Type, Boolean)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
GetHashCode()	Returns the hash code for this instance.
GetType()	Gets the Type of the current instance.

	(Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class.
IsDefined(Assembly, Type)	Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.
IsDefined(Assembly, Type, Boolean)	Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.
IsDefined(MemberInfo, Type)	Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
IsDefined(MemberInfo, Type, Boolean)	Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
IsDefined(Module, Type)	Determines whether any custom attributes of a specified type are applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
IsDefined(Module, Type, Boolean)	Determines whether any custom attributes are applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
IsDefined(ParameterInfo, Type)	Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.
IsDefined(ParameterInfo, Type, Boolean)	Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

This type is thread safe.

See also

- Applying Attributes
- Extending Metadata Using Attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Attribute class.

C#

```
protected Attribute ();
```

Examples

The following code example shows the definition of a custom parameter Attribute class with its constructor.

C#

```
// Define a custom parameter attribute that takes a single message argument.
[AttributeUsage( AttributeTargets.Parameter )]
public class ArgumentUsageAttribute : Attribute
{
    // This is the attribute constructor.
    public ArgumentUsageAttribute( string UsageMsg )
    {
        this.usageMsg = UsageMsg;
    }

    // usageMsg is storage for the attribute message.
    protected string usageMsg;

    // This is the Message property for the attribute.
    public string Message
    {
        get { return usageMsg; }
        set { usageMsg = value; }
    }
}
```

F#

```
// Define a custom parameter attribute that takes a single message argument.
[<AttributeUsage(AttributeTargets.Parameter); AllowNullLiteral>]
type ArgumentUsageAttribute(usageMsg) =
    inherit Attribute()

    // This is the Message property for the attribute.
    member val Message = usageMsg with get, set
```

VB

```
' Define a custom parameter attribute that takes a single message argument.
<AttributeUsage(AttributeTargets.Parameter)>  _
Public Class ArgumentUsageAttribute
    Inherits Attribute

    ' This is the attribute constructor.
    Public Sub New(UsageMsg As String)
        Me.usageMsg = UsageMsg
    End Sub

    ' usageMsg is storage for the attribute message.
    Protected usageMsg As String

    ' This is the Message property for the attribute.
    Public Property Message() As String
        Get
            Return usageMsg
        End Get
        Set
            usageMsg = value
        End Set
    End Property
End Class
```

Remarks

This constructor is only called by classes that derive from Attribute.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Attribute.TypeId Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When implemented in a derived class, gets a unique identifier for this Attribute.

C#

```
public virtual object TypeId { get; }
```

Property Value

Object

An Object that is a unique identifier for the attribute.

Examples

The following code example implements the TypeId property in a custom parameter Attribute class and shows its use.

C#

```
// Example for the Attribute.TypeId property.
using System;
using System.Reflection;

namespace NDP_UE_CS
{
    // Define a custom parameter attribute that takes a single message
argument.
    [AttributeUsage( AttributeTargets.Parameter )]
    public class ArgumentUsageAttribute : Attribute
    {
        // The constructor saves the message and creates a unique
identifier.
        public ArgumentUsageAttribute( string UsageMsg )
        {
            this.usageMsg = UsageMsg;
            this.instanceGUID = Guid.NewGuid( );
```

```
        }

        // This is storage for the attribute message and unique ID.
        protected string usageMsg;
        protected Guid instanceGUID;

        // This is the Message property for the attribute.
        public string Message
        {
            get { return usageMsg; }
            set { usageMsg = value; }
        }

        // Override TypeId to provide a unique identifier for the instance.
        public override object TypeId
        {
            get { return (object)instanceGUID; }
        }

        // Override ToString() to append the message to what the base
generates.
        public override string ToString( )
        {
            return base.ToString( ) + ":" + usageMsg;
        }
    }

    public class TestClass
    {
        // Assign an ArgumentUsage attribute to each parameter.
        // Assign a ParamArray attribute to strList using the params
keyword.
        public void TestMethod(
            [ArgumentUsage("Must pass an array here.")]
            String[] strArray,
            [ArgumentUsage("Can pass a param list or array here.")]
            params String[] strList)
        { }
    }

    class AttributeTypeIdDemo
    {
        static void ShowAttributeTypeIds( )
        {
            // Get the class type, and then get the MethodInfo object
            // for TestMethod to access its metadata.
            Type clsType = typeof( TestClass );
            MethodInfo mInfo = clsType.GetMethod("TestMethod");

            // There will be two elements in pInfoArray, one for each
parameter.
            ParameterInfo[] pInfoArray = mInfo.GetParameters();
            if (pInfoArray != null)
            {
                // Create an instance of the param array attribute on
```

```
strList.
                ParamArrayAttribute listArrayAttr = (ParamArrayAttribute)
                    Attribute.GetCustomAttribute( pInfoArray[1],
                        typeof(ParamArrayAttribute) );

                // Create an instance of the argument usage attribute on
strArray.
                ArgumentUsageAttribute arrayUsageAttr1 =
(ArgumentUsageAttribute)
                    Attribute.GetCustomAttribute( pInfoArray[0],
                        typeof(ArgumentUsageAttribute) );

                // Create another instance of the argument usage attribute
                // on strArray.
                ArgumentUsageAttribute arrayUsageAttr2 =
(ArgumentUsageAttribute)
                    Attribute.GetCustomAttribute( pInfoArray[0],
                        typeof(ArgumentUsageAttribute) );

                // Create an instance of the argument usage attribute on
strList.
                ArgumentUsageAttribute listUsageAttr =
(ArgumentUsageAttribute)
                    Attribute.GetCustomAttribute( pInfoArray[1],
                        typeof(ArgumentUsageAttribute) );

                // Display the attributes and corresponding TypeId values.
                Console.WriteLine( "\n\"{0}\" \nTypeId: {1}",
                    listArrayAttr.ToString(), listArrayAttr.TypeId );
                Console.WriteLine( "\n\"{0}\" \nTypeId: {1}",
                    arrayUsageAttr1.ToString(), arrayUsageAttr1.TypeId );
                Console.WriteLine( "\n\"{0}\" \nTypeId: {1}",
                    arrayUsageAttr2.ToString(), arrayUsageAttr2.TypeId );
                Console.WriteLine( "\n\"{0}\" \nTypeId: {1}",
                    listUsageAttr.ToString(), listUsageAttr.TypeId );
            }
            else
                Console.WriteLine( "The parameters information could " +
                    "not be retrieved for method {0}.", mInfo.Name );
        }

        static void Main( )
        {
            Console.WriteLine(
                "This example of the Attribute.TypeId property\n" +
                "generates the following output." );
            Console.WriteLine(
                "\nCreate instances from a derived Attribute " +
                "class that implements TypeId, \nand then " +
                "display the attributes and corresponding TypeId values:" );

            ShowAttributeTypeIds( );
        }
    }
}
```

```
/*
This example of the Attribute.TypeId property
generates output similar to the following:

Create instances from a derived Attribute class that implements TypeId,
and then display the attributes and corresponding TypeId values:

"System.ParamArrayAttribute"
TypeId: System.ParamArrayAttribute

"NDP_UE_CS.ArgumentUsageAttribute:Must pass an array here."
TypeId: d03a23f4-2536-4478-920f-8b0426dec7f1

"NDP_UE_CS.ArgumentUsageAttribute:Must pass an array here."
TypeId: a1b412e8-3047-49fa-8d03-7660d37ef718

"NDP_UE_CS.ArgumentUsageAttribute:Can pass a param list or array here."
TypeId: 7ac2bf61-0327-48d6-a07e-eb9aaf3dd45e
*/
```

Remarks

As implemented, this identifier is merely the Type of the attribute. However, it is intended that the unique identifier be used to identify two attributes of the same type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Attribute.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An Object to compare with this instance or `null`.

Returns

Boolean

`true` if `obj` and this instance are of the same type and have identical field values; otherwise, `false`.

Examples

The following example defines two custom parameter Attribute classes, then creates several objects of each class and uses the Equals method to compare them.

C#

```
using System;
using System.Reflection;

// Define a custom parameter attribute that takes a single message argument.
[AttributeUsage( AttributeTargets.Parameter )]
public class ArgumentUsageAttribute : Attribute
{
    // This is the attribute constructor.
    public ArgumentUsageAttribute( string UsageMsg )
```

```
    {
        this.usageMsg = UsageMsg;
    }

    // usageMsg is storage for the attribute message.
    protected string usageMsg;

    // Override ToString() to append the message to what the base generates.
    public override string ToString( )
    {
        return $"{base.ToString()}: {usageMsg}";
    }
}

// Define a custom parameter attribute that generates a GUID for each
instance.
[AttributeUsage( AttributeTargets.Parameter )]
public class ArgumentIDAttribute : Attribute
{
    // This is the attribute constructor, which generates the GUID.
    public ArgumentIDAttribute( )
    {
        this.instanceGUID = Guid.NewGuid( );
    }

    // instanceGUID is storage for the generated GUID.
    protected Guid instanceGUID;

    // Override ToString() to append the GUID to what the base generates.
    public override string ToString( )
    {
        return $"{base.ToString()}.{instanceGUID.ToString()}";
    }
}

public class TestClass
{
    // Assign an ArgumentID attribute to each parameter.
    // Assign an ArgumentUsage attribute to each parameter.
    public void TestMethod(
        [ArgumentID]
        [ArgumentUsage("Must pass an array here.")]
        String[] strArray,
        [ArgumentID]
        [ArgumentUsage("Can pass param list or array here.")]
        params String[] strList)
    { }
}

class AttributeEqualsDemo
{
    // Create Attribute objects and compare them.
    static void Main()
    {
        // Get the class type, and then get the MethodInfo object
```

```
        // for TestMethod to access its metadata.
        Type clsType = typeof( TestClass );
        MethodInfo mInfo = clsType.GetMethod("TestMethod");

        // There will be two elements in pInfoArray, one for each parameter.
        ParameterInfo[] pInfoArray = mInfo.GetParameters();
        if (pInfoArray != null)
        {
            // Create an instance of the argument usage attribute on
strArray.
            ArgumentUsageAttribute arrayUsageAttr1 =
(ArgumentUsageAttribute)
                Attribute.GetCustomAttribute( pInfoArray[0],
                    typeof(ArgumentUsageAttribute) );

            // Create another instance of the argument usage attribute
            // on strArray.
            ArgumentUsageAttribute arrayUsageAttr2 =
(ArgumentUsageAttribute)
                Attribute.GetCustomAttribute( pInfoArray[0],
                    typeof(ArgumentUsageAttribute) );

            // Create an instance of the argument usage attribute on
strList.
            ArgumentUsageAttribute listUsageAttr = (ArgumentUsageAttribute)
                Attribute.GetCustomAttribute( pInfoArray[1],
                    typeof(ArgumentUsageAttribute) );

            // Create an instance of the argument ID attribute on strArray.
            ArgumentIDAttribute arrayIDAttr1 = (ArgumentIDAttribute)
                Attribute.GetCustomAttribute( pInfoArray[0],
                    typeof(ArgumentIDAttribute) );

            // Create another instance of the argument ID attribute on
strArray.
            ArgumentIDAttribute arrayIDAttr2 = (ArgumentIDAttribute)
                Attribute.GetCustomAttribute( pInfoArray[0],
                    typeof(ArgumentIDAttribute) );

            // Create an instance of the argument ID attribute on strList.
            ArgumentIDAttribute listIDAttr = (ArgumentIDAttribute)
                Attribute.GetCustomAttribute( pInfoArray[1],
                    typeof(ArgumentIDAttribute) );

            // Compare various pairs of attributes for equality.
            Console.WriteLine("\nCompare a usage attribute instance to " +
                "another instance of the same attribute:" );
            Console.WriteLine($"   \"{arrayUsageAttr1}\" == \n" +
                    $"   \"{arrayUsageAttr2}\"? {arrayUsageAttr1.Equals(
arrayUsageAttr2)}");

            Console.WriteLine("\nCompare a usage attribute to " +
                "another usage attribute:" );
            Console.WriteLine($"   \"{arrayUsageAttr1}\" == \n" +
                    $"   \"{listUsageAttr}\"?
```

```
{arrayUsageAttr1.Equals(listUsageAttr)}");

            Console.WriteLine("\nCompare an ID attribute instance to " +
                "another instance of the same attribute:" );
            Console.WriteLine($"   \"{ arrayIDAttr1}\" == \n" +
                    $"   \"{arrayIDAttr2}\"?
{arrayIDAttr1.Equals(arrayIDAttr2)}");

            Console.WriteLine("\nCompare an ID attribute to another ID
attribute:" );
            Console.WriteLine($"   \"{arrayIDAttr1}\" == \n" +
                    $"   \"{listIDAttr}\"?
{arrayIDAttr1.Equals(listIDAttr)}");
        }
        else
            Console.WriteLine( "The parameters information could " +
                "not be retrieved for method {0}.", mInfo.Name);
    }
    }
/*
The example displays an output similar to the following:
//         Compare a usage attribute instance to another instance of the
same attribute:
//            "ArgumentUsageAttribute: Must pass an array here." ==
//            "ArgumentUsageAttribute: Must pass an array here."? True
//
//         Compare a usage attribute to another usage attribute:
//            "ArgumentUsageAttribute: Must pass an array here." ==
//            "ArgumentUsageAttribute: Can pass param list or array here."?
False
//
//         Compare an ID attribute instance to another instance of the same
attribute:
//            "ArgumentIDAttribute.22d1a176-4aca-427b-8230-0c1563e13187" ==
//            "ArgumentIDAttribute.7fa94bba-c290-48e1-a0de-e22f6c1e64f1"?
False
//
//         Compare an ID attribute to another ID attribute:
//            "ArgumentIDAttribute.22d1a176-4aca-427b-8230-0c1563e13187" ==
//            "ArgumentIDAttribute.b9eea70d-9c0f-459e-a984-19c46b6c8789"?
False
*/
```

Remarks

The Equals method uses reflection to retrieve field information for the current instance and the `obj` argument. It then compares field values.

When implementing your own class derived from Attribute, you can override the Equals method. Because it uses reflection, we recommend that you do so. However, your override should perform a standard test for reference equality (the two arguments

represent the same object instance) or value equality (the two arguments are of the same type and have identical field values). If you want to perform a custom comparison to determine whether two attributes objects are equal, you can override the Match method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute.GetCustomAttribute Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves a custom attribute of a specified type applied to an assembly, module, type member, or method parameter.

Overloads

Expand table

GetCustomAttribute(ParameterInfo, Type, Boolean)	Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
GetCustomAttribute(MemberInfo, Type, Boolean)	Retrieves a custom attribute applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttribute(Assembly, Type, Boolean)	Retrieves a custom attribute applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttribute(Module, Type, Boolean)	Retrieves a custom attribute applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttribute(Module, Type)	Retrieves a custom attribute applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
GetCustomAttribute(MemberInfo, Type)	Retrieves a custom attribute applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
GetCustomAttribute(Assembly, Type)	Retrieves a custom attribute applied to a specified assembly. Parameters specify the assembly and the type of the custom attribute to search for.

GetCustomAttribute(ParameterInfo, Type)	Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.

GetCustomAttribute(ParameterInfo, Type, Boolean)

Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.

C#

```
public static Attribute? GetCustomAttribute
(System.Reflection.ParameterInfo element, Type attributeType, bool
inherit);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example defines a custom parameter Attribute class and applies the custom attribute to a method in a derived class and the base of the derived class. The example shows the use of the GetCustomAttribute method to return the attributes.

C#

```
// Example for the Attribute.GetCustomAttribute( ParameterInfo, Type,
Boolean )
// method.
using System;
using System.Reflection;

namespace NDP_UE_CS
{
    // Define a custom parameter attribute that takes a single message
argument.
    [AttributeUsage( AttributeTargets.Parameter )]
    public class ArgumentUsageAttribute : Attribute
    {
        // This is the attribute constructor.
        public ArgumentUsageAttribute( string UsageMsg )
        {
            this.usageMsg = UsageMsg;
        }

        // usageMsg is storage for the attribute message.
        protected string usageMsg;

        // This is the Message property for the attribute.
        public string Message
        {
            get { return usageMsg; }
            set { usageMsg = value; }
        }
    }

    public class BaseClass
```

```
    {
        // Assign an ArgumentUsage attribute to the strArray parameter.
        // Assign a ParamArray attribute to strList using the params
keyword.
        public virtual void TestMethod(
            [ArgumentUsage("Must pass an array here.")]
            String[] strArray,
            params String[] strList)
        { }
    }

    public class DerivedClass : BaseClass
    {
        // Assign an ArgumentUsage attribute to the strList parameter.
        // Assign a ParamArray attribute to strList using the params
keyword.
        public override void TestMethod(
            String[] strArray,
            [ArgumentUsage("Can pass a parameter list or array here.")]
            params String[] strList)
        { }
    }

    class CustomParamDemo
    {
        static void Main( )
        {
            Console.WriteLine(
                "This example of Attribute.GetCustomAttribute( Parameter"
+
                "Info, Type, Boolean )\ngenerates the following output."
);

            // Get the class type, and then get the MethodInfo object
            // for TestMethod to access its metadata.
            Type clsType = typeof(DerivedClass);
            MethodInfo mInfo = clsType.GetMethod("TestMethod");

            // Iterate through the ParameterInfo array for the method
parameters.
            ParameterInfo[] pInfoArray = mInfo.GetParameters();
            if (pInfoArray != null)
            {
                DisplayParameterAttributes( mInfo, pInfoArray, false );
                DisplayParameterAttributes( mInfo, pInfoArray, true );
            }
            else
                Console.WriteLine("The parameters information could " +
                    "not be retrieved for method {0}.", mInfo.Name);
        }

        static void DisplayParameterAttributes( MethodInfo mInfo,
            ParameterInfo[] pInfoArray, bool includeInherited )
        {
            Console.WriteLine(
```

```
                "\nParameter attribute information for method \"" +
                "{0}\"\nincludes inheritance from base class: {1}.",
                mInfo.Name, includeInherited ? "Yes" : "No" );

            // Display the attribute information for the parameters.
            foreach( ParameterInfo paramInfo in pInfoArray )
            {
                // See if the ParamArray attribute is defined.
                bool isDef = Attribute.IsDefined( paramInfo,
                    typeof(ParamArrayAttribute));

                if( isDef )
                    Console.WriteLine(
                        "\n    The ParamArray attribute is defined " +
                        "for \n    parameter {0} of method {1}.",
                        paramInfo.Name, mInfo.Name);

                // See if ParamUsageAttribute is defined.
                // If so, display a message.
                ArgumentUsageAttribute usageAttr =
(ArgumentUsageAttribute)
                    Attribute.GetCustomAttribute( paramInfo,
                        typeof(ArgumentUsageAttribute),
                        includeInherited );

                if( usageAttr != null )
                {
                    Console.WriteLine(
                        "\n    The ArgumentUsage attribute is def" +
                        "ined for \n    parameter {0} of method {1}.",
                        paramInfo.Name, mInfo.Name );

                    Console.WriteLine( "\n        The usage " +
                        "message for {0} is:\n        \"{1}\".",
                        paramInfo.Name, usageAttr.Message);
                }
            }
        }
    }
}

/*
This example of Attribute.GetCustomAttribute( ParameterInfo, Type,
Boolean )
generates the following output.

Parameter attribute information for method "TestMethod"
includes inheritance from base class: No.

    The ParamArray attribute is defined for
    parameter strList of method TestMethod.

    The ArgumentUsage attribute is defined for
    parameter strList of method TestMethod.
```

```
        The usage message for strList is:
        "Can pass a parameter list or array here.".

Parameter attribute information for method "TestMethod"
includes inheritance from base class: Yes.

    The ArgumentUsage attribute is defined for
    parameter strArray of method TestMethod.

        The usage message for strArray is:
        "Must pass an array here.".

    The ParamArray attribute is defined for
    parameter strList of method TestMethod.

    The ArgumentUsage attribute is defined for
    parameter strList of method TestMethod.

        The usage message for strList is:
        "Can pass a parameter list or array here.".
*/
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(MemberInfo, Type, Boolean)

Retrieves a custom attribute applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.

```
C#

public static Attribute? GetCustomAttribute (System.Reflection.MemberInfo
element, Type attributeType, bool inherit);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

AmbiguousMatchException

More than one of the requested attributes was found.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;

namespace IsDef4CS
{
    public class TestClass
    {
        // Assign the Obsolete attribute to a method.
        [Obsolete("This method is obsolete. Use Method2 instead.")]
        public void Method1()
        {}
        public void Method2()
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // See if the Obsolete attribute is defined for this method.
            bool isDef = Attribute.IsDefined(mInfo,
typeof(ObsoleteAttribute));
            // Display the result.
            Console.WriteLine("The Obsolete Attribute {0} defined for {1}
of class {2}.",
                isDef ? "is" : "is not", mInfo.Name, clsType.Name);
            // If it's defined, display the attribute's message.
            if (isDef)
            {
                ObsoleteAttribute obsAttr =

(ObsoleteAttribute)Attribute.GetCustomAttribute(
                                                        mInfo,
typeof(ObsoleteAttribute));
                if (obsAttr != null)
                    Console.WriteLine("The message is: \"{0}\".",
                        obsAttr.Message);
                else
```

```
                    Console.WriteLine("The message could not be
retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Obsolete Attribute is defined for Method1 of class TestClass.
 * The message is: "This method is obsolete. Use Method2 instead.".
 */
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on types, methods, and constructors if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(Assembly, Type, Boolean)

Retrieves a custom attribute applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.

```
C#

public static Attribute? GetCustomAttribute (System.Reflection.Assembly
element, Type attributeType, bool inherit);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking an Assembly as a parameter.

```
C#
```

```
using System;
using System.Reflection;

// Add an AssemblyDescription attribute
[assembly: AssemblyDescription("A sample description")]
namespace IsDef1CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // Get the assembly object.
            Assembly assy = clsType.Assembly;
            // Store the assembly's name.
            String assyName = assy.GetName().Name;
            // See if the Assembly Description is defined.
            bool isdef = Attribute.IsDefined(assy,
                typeof(AssemblyDescriptionAttribute));
            if (isdef)
            {
                // Affirm that the attribute is defined.
                Console.WriteLine("The AssemblyDescription attribute " +
                    "is defined for assembly {0}.", assyName);
                // Get the description attribute itself.
                AssemblyDescriptionAttribute adAttr =

(AssemblyDescriptionAttribute)Attribute.GetCustomAttribute(
                    assy, typeof(AssemblyDescriptionAttribute));
                // Display the description.
                if (adAttr != null)
                    Console.WriteLine("The description is \"{0}\".",
                        adAttr.Description);
                else
                    Console.WriteLine("The description could not " +
                        "be retrieved.");
            }
            else
                Console.WriteLine("The AssemblyDescription attribute is
not " +
                    "defined for assembly {0}.", assyName);
        }
    }
}

/*
 * Output:
 * The AssemblyDescription attribute is defined for assembly IsDef1CS.
 * The description is "A sample description".
 */
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(Module, Type, Boolean)

Retrieves a custom attribute applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.

C#

```
public static Attribute? GetCustomAttribute (System.Reflection.Module
element, Type attributeType, bool inherit);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking a Module as a parameter.

C#

```
using System;
using System.Diagnostics;

// Add the Debuggable attribute to the module.
[module:Debuggable(true, false)]
namespace IsDef2CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // See if the Debuggable attribute is defined for this
module.
            bool isDef = Attribute.IsDefined(clsType.Module,
                typeof(DebuggableAttribute));
            // Display the result.
```

```
            Console.WriteLine("The Debuggable attribute {0} " +
                "defined for Module {1}.",
                isDef ? "is" : "is not",
                clsType.Module.Name);
            // If the attribute is defined, display the JIT settings.
            if (isDef)
            {
                // Retrieve the attribute itself.
                DebuggableAttribute dbgAttr = (DebuggableAttribute)
                    Attribute.GetCustomAttribute(clsType.Module,
                    typeof(DebuggableAttribute));
                if (dbgAttr != null)
                {
                    Console.WriteLine("JITTrackingEnabled is {0}.",
                        dbgAttr.IsJITTrackingEnabled);
                    Console.WriteLine("JITOptimizerDisabled is {0}.",
                        dbgAttr.IsJITOptimizerDisabled);
                }
                else
                    Console.WriteLine("The Debuggable attribute " +
                        "could not be retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Debuggable attribute is defined for Module IsDef2CS.exe.
 * JITTrackingEnabled is True.
 * JITOptimizerDisabled is False.
 */
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(Module, Type)

Retrieves a custom attribute applied to a module. Parameters specify the module, and the type of the custom attribute to search for.

```C#
public static Attribute? GetCustomAttribute (System.Reflection.Module element, Type attributeType);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking a Module as a parameter.

```C#
```

```
using System;
using System.Diagnostics;

// Add the Debuggable attribute to the module.
[module:Debuggable(true, false)]
namespace IsDef2CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // See if the Debuggable attribute is defined for this
module.
            bool isDef = Attribute.IsDefined(clsType.Module,
                typeof(DebuggableAttribute));
            // Display the result.
            Console.WriteLine("The Debuggable attribute {0} " +
                "defined for Module {1}.",
                isDef ? "is" : "is not",
                clsType.Module.Name);
            // If the attribute is defined, display the JIT settings.
            if (isDef)
            {
                // Retrieve the attribute itself.
                DebuggableAttribute dbgAttr = (DebuggableAttribute)
                    Attribute.GetCustomAttribute(clsType.Module,
                    typeof(DebuggableAttribute));
                if (dbgAttr != null)
                {
                    Console.WriteLine("JITTrackingEnabled is {0}.",
                        dbgAttr.IsJITTrackingEnabled);
                    Console.WriteLine("JITOptimizerDisabled is {0}.",
                        dbgAttr.IsJITOptimizerDisabled);
                }
                else
                    Console.WriteLine("The Debuggable attribute " +
                        "could not be retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Debuggable attribute is defined for Module IsDef2CS.exe.
 * JITTrackingEnabled is True.
 * JITOptimizerDisabled is False.
 */
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(MemberInfo, Type)

Retrieves a custom attribute applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.

C#

```
public static Attribute? GetCustomAttribute (System.Reflection.MemberInfo
element, Type attributeType);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

AmbiguousMatchException

More than one of the requested attributes was found.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;

namespace IsDef4CS
{
    public class TestClass
    {
        // Assign the Obsolete attribute to a method.
        [Obsolete("This method is obsolete. Use Method2 instead.")]
        public void Method1()
        {}
        public void Method2()
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // See if the Obsolete attribute is defined for this method.
            bool isDef = Attribute.IsDefined(mInfo,
typeof(ObsoleteAttribute));
            // Display the result.
            Console.WriteLine("The Obsolete Attribute {0} defined for {1}
of class {2}.",
                isDef ? "is" : "is not", mInfo.Name, clsType.Name);
            // If it's defined, display the attribute's message.
```

```
            if (isDef)
            {
                ObsoleteAttribute obsAttr =

(ObsoleteAttribute)Attribute.GetCustomAttribute(
                                                        mInfo,
typeof(ObsoleteAttribute));
                if (obsAttr != null)
                    Console.WriteLine("The message is: \"{0}\".",
                        obsAttr.Message);
                else
                    Console.WriteLine("The message could not be
retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Obsolete Attribute is defined for Method1 of class TestClass.
 * The message is: "This method is obsolete. Use Method2 instead.".
 */
```

Remarks

A match is determined in the same way described in the Return Value section of Type.IsAssignableFrom.

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on types, methods, and constructors if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(Assembly, Type)

Retrieves a custom attribute applied to a specified assembly. Parameters specify the assembly and the type of the custom attribute to search for.

```
C#

public static Attribute? GetCustomAttribute (System.Reflection.Assembly
element, Type attributeType);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

Examples

The following code example illustrates the use of the GetCustomAttribute method taking an Assembly as a parameter.

C#

```
using System;
using System.Reflection;

// Add an AssemblyDescription attribute
[assembly: AssemblyDescription("A sample description")]
namespace IsDef1CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // Get the assembly object.
            Assembly assy = clsType.Assembly;
            // Store the assembly's name.
            String assyName = assy.GetName().Name;
            // See if the Assembly Description is defined.
            bool isdef = Attribute.IsDefined(assy,
                typeof(AssemblyDescriptionAttribute));
            if (isdef)
            {
                // Affirm that the attribute is defined.
                Console.WriteLine("The AssemblyDescription attribute " +
                    "is defined for assembly {0}.", assyName);
                // Get the description attribute itself.
                AssemblyDescriptionAttribute adAttr =

(AssemblyDescriptionAttribute)Attribute.GetCustomAttribute(
                    assy, typeof(AssemblyDescriptionAttribute));
                // Display the description.
                if (adAttr != null)
                    Console.WriteLine("The description is \"{0}\".",
                        adAttr.Description);
                else
                    Console.WriteLine("The description could not " +
                        "be retrieved.");
            }
            else
                Console.WriteLine("The AssemblyDescription attribute is
not " +
                    "defined for assembly {0}.", assyName);
        }
```

```
    }
}

/*
 * Output:
 * The AssemblyDescription attribute is defined for assembly IsDef1CS.
 * The description is "A sample description".
 */
```

Remarks

Use the GetCustomAttributes method if you expect more than one value to be returned, or AmbiguousMatchException will be thrown.

> ⓘ **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttribute(ParameterInfo, Type)

Retrieves a custom attribute applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.

C#

```
public static Attribute? GetCustomAttribute
(System.Reflection.ParameterInfo element, Type attributeType);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute

A reference to the single custom attribute of type `attributeType` that is applied to `element`, or `null` if there is no such attribute.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

AmbiguousMatchException

More than one of the requested attributes was found.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example defines a custom parameter Attribute class and applies the custom attribute to a method in a derived class and the base of the derived class. The example shows the use of the GetCustomAttribute method to return the attributes.

C#

```
// Example for the Attribute.GetCustomAttribute( ParameterInfo, Type )
method.
using System;
using System.Reflection;

namespace NDP_UE_CS
{
    // Define a custom parameter attribute that takes a single message
argument.
    [AttributeUsage( AttributeTargets.Parameter )]
    public class ArgumentUsageAttribute : Attribute
    {
        // This is the attribute constructor.
        public ArgumentUsageAttribute( string UsageMsg )
        {
            this.usageMsg = UsageMsg;
        }

        // usageMsg is storage for the attribute message.
        protected string usageMsg;

        // This is the Message property for the attribute.
        public string Message
        {
            get { return usageMsg; }
            set { usageMsg = value; }
        }
    }

    public class BaseClass
    {
        // Assign an ArgumentUsage attribute to the strArray parameter.
        // Assign a ParamArray attribute to strList using the params
keyword.
        public virtual void TestMethod(
            [ArgumentUsage("Must pass an array here.")]
            String[] strArray,
            params String[] strList)
        { }
    }

    public class DerivedClass : BaseClass
    {
        // Assign an ArgumentUsage attribute to the strList parameter.
        // Assign a ParamArray attribute to strList using the params
keyword.
        public override void TestMethod(
            String[] strArray,
            [ArgumentUsage("Can pass a parameter list or array here.")]
            params String[] strList)
        { }
    }
```

```
    class CustomParamDemo
    {
        static void Main( )
        {
            Console.WriteLine(
                "This example of Attribute.GetCustomAttribute( Param" +
                "eterInfo, Type )\ngenerates the following output." );

            // Get the class type, and then get the MethodInfo object
            // for TestMethod to access its metadata.
            Type clsType = typeof( DerivedClass );
            MethodInfo mInfo = clsType.GetMethod("TestMethod");

            // Iterate through the ParameterInfo array for the method
parameters.
            ParameterInfo[] pInfoArray = mInfo.GetParameters();
            if (pInfoArray != null)
            {
                foreach( ParameterInfo paramInfo in pInfoArray )
                {
                    // See if the ParamArray attribute is defined.
                    bool isDef = Attribute.IsDefined(
                        paramInfo, typeof(ParamArrayAttribute));

                    if( isDef )
                        Console.WriteLine(
                            "\nThe ParamArray attribute is defined " +
                            "for \nparameter {0} of method {1}.",
                            paramInfo.Name, mInfo.Name);

                    // See if ParamUsageAttribute is defined.
                    // If so, display a message.
                    ArgumentUsageAttribute usageAttr =
(ArgumentUsageAttribute)
                        Attribute.GetCustomAttribute(
                            paramInfo, typeof(ArgumentUsageAttribute) );

                    if( usageAttr != null )
                    {
                        Console.WriteLine(
                            "\nThe ArgumentUsage attribute is defined " +
                            "for \nparameter {0} of method {1}.",
                            paramInfo.Name, mInfo.Name );

                        Console.WriteLine( "\n    The usage " +
                            "message for {0} is:\n    \"{1}\".",
                            paramInfo.Name, usageAttr.Message);
                    }
                }
            }
            else
                Console.WriteLine(
                    "The parameters information could not " +
                    "be retrieved for method {0}.", mInfo.Name);
```

```
        }
    }
}

/*
This example of Attribute.GetCustomAttribute( ParameterInfo, Type )
generates the following output.

The ArgumentUsage attribute is defined for
parameter strArray of method TestMethod.

    The usage message for strArray is:
    "Must pass an array here.".

The ParamArray attribute is defined for
parameter strList of method TestMethod.

The ArgumentUsage attribute is defined for
parameter strList of method TestMethod.

    The usage message for strList is:
    "Can pass a parameter list or array here.".
*/
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Attribute.GetCustomAttributes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves an array of the custom attributes applied to an assembly, module, type member, or method parameter.

Overloads

Expand table

GetCustomAttributes(ParameterInfo, Boolean)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and whether to search ancestors of the method parameter.
GetCustomAttributes(MemberInfo, Type, Boolean)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttributes(ParameterInfo, Type, Boolean)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
GetCustomAttributes(Module, Type, Boolean)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
GetCustomAttributes(MemberInfo, Type)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
GetCustomAttributes(Assembly, Type, Boolean)	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.

GetCustomAttributes(Module, Type)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
GetCustomAttributes(ParameterInfo, Type)	Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.
GetCustomAttributes(MemberInfo, Boolean)	Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
GetCustomAttributes(Assembly, Type)	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.
GetCustomAttributes(Assembly, Boolean)	Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and an ignored search option.
GetCustomAttributes(ParameterInfo)	Retrieves an array of the custom attributes applied to a method parameter. A parameter specifies the method parameter.
GetCustomAttributes(Module)	Retrieves an array of the custom attributes applied to a module. A parameter specifies the module.
GetCustomAttributes(MemberInfo)	Retrieves an array of the custom attributes applied to a member of a type. A parameter specifies the member.
GetCustomAttributes(Assembly)	Retrieves an array of the custom attributes applied to an assembly. A parameter specifies the assembly.
GetCustomAttributes(Module, Boolean)	Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and an ignored search option.

GetCustomAttributes(ParameterInfo, Boolean)

Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and whether to search ancestors of the method parameter.

```
C#
```

```
public static Attribute[] GetCustomAttributes
(System.Reflection.ParameterInfo element, bool inherit);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentException

The Member property of `element` is `null`.

ArgumentNullException

`element` is `null`.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a ParameterInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

namespace CustAttrs5CS {
```

```
    public class AClass {
        public void ParamArrayAndDesc(
            // Add ParamArray (with the keyword) and Description
attributes.
            [Description("This argument is a ParamArray")]
            params int[] args)
        {}
    }

    class DemoClass {
        static void Main(string[] args) {
            // Get the Class type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for the method.
            MethodInfo mInfo = clsType.GetMethod("ParamArrayAndDesc");
            if (mInfo != null) {
                // Get the parameter information.
                ParameterInfo[] pInfo = mInfo.GetParameters();
                if (pInfo != null) {
                    // Iterate through all the attributes for the
parameter.
                    foreach(Attribute attr in
                        Attribute.GetCustomAttributes(pInfo[0])) {
                        // Check for the ParamArray attribute.
                        if (attr.GetType() ==
typeof(ParamArrayAttribute))
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has the ParamArray attribute.",
                                pInfo[0].Name, mInfo.Name);
                        // Check for the Description attribute.
                        else if (attr.GetType() ==
                            typeof(DescriptionAttribute)) {
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has a description attribute.",
                                pInfo[0].Name, mInfo.Name);
                            Console.WriteLine("The description is: \"
{0}\"",

((DescriptionAttribute)attr).Description);
                        }
                    }
                }
            }
        }
    }
}

/*
 * Output:
 * Parameter args for method ParamArrayAndDesc has the ParamArray
attribute.
 * Parameter args for method ParamArrayAndDesc has a description
attribute.
```

```
  * The description is: "This argument is a ParamArray"
  */
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(MemberInfo, Type, Boolean)

Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.

C#

```
public static Attribute[] GetCustomAttributes
(System.Reflection.MemberInfo element, Type attributeType, bool inherit);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `type` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `type` is `null`.

ArgumentException

`type` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.Security;
using System.Runtime.InteropServices;

namespace CustAttrs4CS
{

    // Define an enumeration of Win32 unmanaged types
    public enum UnmanagedType
    {
```

```
        User,
        GDI,
        Kernel,
        Shell,
        Networking,
        Multimedia
    }

    // Define the Unmanaged attribute.
    public class UnmanagedAttribute : Attribute
    {
        // Storage for the UnmanagedType value.
        protected UnmanagedType thisType;

        // Set the unmanaged type in the constructor.
        public UnmanagedAttribute(UnmanagedType type)
        {
            thisType = type;
        }

        // Define a property to get and set the UnmanagedType value.
        public UnmanagedType Win32Type
        {
            get { return thisType; }
            set { thisType = Win32Type; }
        }
    }

    // Create a class for an imported Win32 unmanaged function.
    public class Win32 {
        [DllImport("user32.dll", CharSet = CharSet.Unicode)]
        public static extern int MessageBox(int hWnd, String text,
            String caption, uint type);
    }

    public class AClass {
        // Add some attributes to Win32CallMethod.
        [Obsolete("This method is obsolete. Use managed MsgBox
instead.")]
        [Unmanaged(UnmanagedType.User)]
        public void Win32CallMethod()
        {
            Win32.MessageBox(0, "This is an unmanaged call.", "Caution!",
0);
        }
    }

    class DemoClass {
        static void Main(string[] args)
            {
            // Get the AClass type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for Win32CallMethod.
            MethodInfo mInfo = clsType.GetMethod("Win32CallMethod");
            if (mInfo != null)
```

```
            {
                // Iterate through all the attributes of the method.
                foreach(Attribute attr in
                    Attribute.GetCustomAttributes(mInfo)) {
                    // Check for the Obsolete attribute.
                    if (attr.GetType() == typeof(ObsoleteAttribute))
                    {
                        Console.WriteLine("Method {0} is obsolete. " +
                            "The message is:",
                            mInfo.Name);
                        Console.WriteLine("  \"{0}\"",
                            ((ObsoleteAttribute)attr).Message);
                    }

                    // Check for the Unmanaged attribute.
                    else if (attr.GetType() ==
typeof(UnmanagedAttribute))
                    {
                        Console.WriteLine(
                            "This method calls unmanaged code.");
                        Console.WriteLine(
                            String.Format("The Unmanaged attribute type
is {0}.",

((UnmanagedAttribute)attr).Win32Type));
                        AClass myCls = new AClass();
                        myCls.Win32CallMethod();
                    }
                }
            }
        }
    }
}

/*

This code example produces the following results.

First, the compilation yields the warning, "... This method is
obsolete. Use managed MsgBox instead."
Second, execution yields a message box with a title of "Caution!"
and message text of "This is an unmanaged call."
Third, the following text is displayed in the console window:

Method Win32CallMethod is obsolete. The message is:
  "This method is obsolete. Use managed MsgBox instead."
This method calls unmanaged code.
The Unmanaged attribute type is User.

*/
```

Remarks

The return value contains the custom attributes for ancestors of `element` if `inherit` is `true`.

> ⓘ **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on methods, constructors, and types if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(ParameterInfo, Type, Boolean)

Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.

C#

```
public static Attribute[] GetCustomAttributes
(System.Reflection.ParameterInfo element, Type attributeType, bool
inherit);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a ParameterInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

namespace CustAttrs5CS {
    public class AClass {
        public void ParamArrayAndDesc(
            // Add ParamArray (with the keyword) and Description
attributes.
            [Description("This argument is a ParamArray")]
```

```
            params int[] args)
        {}
    }

    class DemoClass {
        static void Main(string[] args) {
            // Get the Class type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for the method.
            MethodInfo mInfo = clsType.GetMethod("ParamArrayAndDesc");
            if (mInfo != null) {
                // Get the parameter information.
                ParameterInfo[] pInfo = mInfo.GetParameters();
                if (pInfo != null) {
                    // Iterate through all the attributes for the
parameter.
                    foreach(Attribute attr in
                        Attribute.GetCustomAttributes(pInfo[0])) {
                        // Check for the ParamArray attribute.
                        if (attr.GetType() ==
typeof(ParamArrayAttribute))
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has the ParamArray attribute.",
                                pInfo[0].Name, mInfo.Name);
                        // Check for the Description attribute.
                        else if (attr.GetType() ==
                            typeof(DescriptionAttribute)) {
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has a description attribute.",
                                pInfo[0].Name, mInfo.Name);
                            Console.WriteLine("The description is: \"
{0}\"",

((DescriptionAttribute)attr).Description);
                        }
                    }
                }
            }
        }
    }
}

/*
 * Output:
 * Parameter args for method ParamArrayAndDesc has the ParamArray
attribute.
 * Parameter args for method ParamArrayAndDesc has a description
attribute.
 * The description is: "This argument is a ParamArray"
 */
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Module, Type, Boolean)

Retrieves an array of the custom attributes applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.

C#

```
public static Attribute[] GetCustomAttributes (System.Reflection.Module
element, Type attributeType, bool inherit);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a Module as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

// Assign some attributes to the module.
[module:Description("A sample description")]

// Set the module's CLSCompliant attribute to false
// The CLSCompliant attribute is applicable for /target:module.
[module:CLSCompliant(false)]

namespace CustAttrs2CS {
    class DemoClass {
        static void Main(string[] args) {
            Type clsType = typeof(DemoClass);
            // Get the Module type to access its metadata.
            Module module = clsType.Module;

            // Iterate through all the attributes for the module.
            foreach(Attribute attr in
Attribute.GetCustomAttributes(module)) {
                // Check for the Description attribute.
                if (attr.GetType() == typeof(DescriptionAttribute))
                    Console.WriteLine("Module {0} has the description " +
                        "\"{1}\".", module.Name,
                        ((DescriptionAttribute)attr).Description);
                // Check for the CLSCompliant attribute.
```

```
                else if (attr.GetType() == typeof(CLSCompliantAttribute))
                    Console.WriteLine("Module {0} {1} CLSCompliant.",
                        module.Name,
                        ((CLSCompliantAttribute)attr).IsCompliant ?
                            "is" : "is not");
            }
        }
    }
}

/*
 * Output:
 * Module CustAttrs2CS.exe is not CLSCompliant.
 * Module CustAttrs2CS.exe has the description "A sample description".
 */
```

Remarks

The return value contains the custom attributes for ancestors of `element` if `inherit` is `true`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(MemberInfo, Type)

Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.

C#

```
public static Attribute[] GetCustomAttributes
(System.Reflection.MemberInfo element, Type attributeType);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `type` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `type` is `null`.

ArgumentException

`type` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttribute, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.Security;
using System.Runtime.InteropServices;

namespace CustAttrs4CS
{
```

```
    // Define an enumeration of Win32 unmanaged types
    public enum UnmanagedType
    {
        User,
        GDI,
        Kernel,
        Shell,
        Networking,
        Multimedia
    }

    // Define the Unmanaged attribute.
    public class UnmanagedAttribute : Attribute
    {
        // Storage for the UnmanagedType value.
        protected UnmanagedType thisType;

        // Set the unmanaged type in the constructor.
        public UnmanagedAttribute(UnmanagedType type)
        {
            thisType = type;
        }

        // Define a property to get and set the UnmanagedType value.
        public UnmanagedType Win32Type
        {
            get { return thisType; }
            set { thisType = Win32Type; }
        }
    }

    // Create a class for an imported Win32 unmanaged function.
    public class Win32 {
        [DllImport("user32.dll", CharSet = CharSet.Unicode)]
        public static extern int MessageBox(int hWnd, String text,
            String caption, uint type);
    }

    public class AClass {
        // Add some attributes to Win32CallMethod.
        [Obsolete("This method is obsolete. Use managed MsgBox
instead.")]
        [Unmanaged(UnmanagedType.User)]
        public void Win32CallMethod()
        {
            Win32.MessageBox(0, "This is an unmanaged call.", "Caution!",
0);
        }
    }

    class DemoClass {
        static void Main(string[] args)
            {
            // Get the AClass type to access its metadata.
```

```
            Type clsType = typeof(AClass);
            // Get the type information for Win32CallMethod.
            MethodInfo mInfo = clsType.GetMethod("Win32CallMethod");
            if (mInfo != null)
            {
                // Iterate through all the attributes of the method.
                foreach(Attribute attr in
                    Attribute.GetCustomAttributes(mInfo)) {
                    // Check for the Obsolete attribute.
                    if (attr.GetType() == typeof(ObsoleteAttribute))
                    {
                        Console.WriteLine("Method {0} is obsolete. " +
                            "The message is:",
                            mInfo.Name);
                        Console.WriteLine("  \"{0}\"",
                            ((ObsoleteAttribute)attr).Message);
                    }

                    // Check for the Unmanaged attribute.
                    else if (attr.GetType() ==
typeof(UnmanagedAttribute))
                    {
                        Console.WriteLine(
                            "This method calls unmanaged code.");
                        Console.WriteLine(
                            String.Format("The Unmanaged attribute type
is {0}.",

((UnmanagedAttribute)attr).Win32Type));
                        AClass myCls = new AClass();
                        myCls.Win32CallMethod();
                    }
                }
            }
        }
    }
}

/*

This code example produces the following results.

First, the compilation yields the warning, "... This method is
obsolete. Use managed MsgBox instead."
Second, execution yields a message box with a title of "Caution!"
and message text of "This is an unmanaged call."
Third, the following text is displayed in the console window:

Method Win32CallMethod is obsolete. The message is:
  "This method is obsolete. Use managed MsgBox instead."
This method calls unmanaged code.
The Unmanaged attribute type is User.

*/
```

Remarks

The return value contains the custom attributes for ancestors of `element`.

> ⓘ **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on methods, constructors, and types if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Assembly, Type, Boolean)

Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.

C#

```
public static Attribute[] GetCustomAttributes (System.Reflection.Assembly
element, Type attributeType, bool inherit);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking an Assembly as a parameter.

C#

```
using System;
using System.Reflection;

[assembly: AssemblyTitle("CustAttrs1CS")]
[assembly: AssemblyDescription("GetCustomAttributes() Demo")]
[assembly: AssemblyCompany("Microsoft")]

class Example {
    static void Main() {
        // Get the Assembly object to access its metadata.
        Assembly assy = typeof(Example).Assembly;

        // Iterate through the attributes for the assembly.
        foreach(Attribute attr in Attribute.GetCustomAttributes(assy)) {
```

```
            // Check for the AssemblyTitle attribute.
            if (attr.GetType() == typeof(AssemblyTitleAttribute))
                Console.WriteLine("Assembly title is \"{0}\".",
                    ((AssemblyTitleAttribute)attr).Title);

            // Check for the AssemblyDescription attribute.
            else if (attr.GetType() ==
                typeof(AssemblyDescriptionAttribute))
                Console.WriteLine("Assembly description is \"{0}\".",
                    ((AssemblyDescriptionAttribute)attr).Description);

            // Check for the AssemblyCompany attribute.
            else if (attr.GetType() == typeof(AssemblyCompanyAttribute))
                Console.WriteLine("Assembly company is {0}.",
                    ((AssemblyCompanyAttribute)attr).Company);
        }
    }
}
// The example displays the following output:
//     Assembly title is "CustAttrs1CS".
//     Assembly description is "GetCustomAttributes() Demo".
//     Assembly company is Microsoft.
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Module, Type)

Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and the type of the custom attribute to search for.

```
C#

public static Attribute[] GetCustomAttributes (System.Reflection.Module
element, Type attributeType);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a Module as a parameter.

```
C#

using System;
using System.Reflection;
```

```
using System.ComponentModel;

// Assign some attributes to the module.
[module:Description("A sample description")]

// Set the module's CLSCompliant attribute to false
// The CLSCompliant attribute is applicable for /target:module.
[module:CLSCompliant(false)]

namespace CustAttrs2CS {
    class DemoClass {
        static void Main(string[] args) {
            Type clsType = typeof(DemoClass);
            // Get the Module type to access its metadata.
            Module module = clsType.Module;

            // Iterate through all the attributes for the module.
            foreach(Attribute attr in
Attribute.GetCustomAttributes(module)) {
                // Check for the Description attribute.
                if (attr.GetType() == typeof(DescriptionAttribute))
                    Console.WriteLine("Module {0} has the description " +
                        "\"{1}\".", module.Name,
                        ((DescriptionAttribute)attr).Description);
                // Check for the CLSCompliant attribute.
                else if (attr.GetType() == typeof(CLSCompliantAttribute))
                    Console.WriteLine("Module {0} {1} CLSCompliant.",
                        module.Name,
                        ((CLSCompliantAttribute)attr).IsCompliant ?
                            "is" : "is not");
            }
        }
    }
}

/*
 * Output:
 * Module CustAttrs2CS.exe is not CLSCompliant.
 * Module CustAttrs2CS.exe has the description "A sample description".
 */
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

GetCustomAttributes(ParameterInfo, Type)

Retrieves an array of the custom attributes applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.

C#

```
public static Attribute[] GetCustomAttributes
(System.Reflection.ParameterInfo element, Type attributeType);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a ParameterInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

namespace CustAttrs5CS {
    public class AClass {
        public void ParamArrayAndDesc(
            // Add ParamArray (with the keyword) and Description
attributes.
            [Description("This argument is a ParamArray")]
            params int[] args)
        {}
    }

    class DemoClass {
        static void Main(string[] args) {
            // Get the Class type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for the method.
            MethodInfo mInfo = clsType.GetMethod("ParamArrayAndDesc");
            if (mInfo != null) {
                // Get the parameter information.
                ParameterInfo[] pInfo = mInfo.GetParameters();
                if (pInfo != null) {
                    // Iterate through all the attributes for the
parameter.
                    foreach(Attribute attr in
                        Attribute.GetCustomAttributes(pInfo[0])) {
                        // Check for the ParamArray attribute.
                        if (attr.GetType() ==
typeof(ParamArrayAttribute))
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has the ParamArray attribute.",
                                pInfo[0].Name, mInfo.Name);
                        // Check for the Description attribute.
                        else if (attr.GetType() ==
                            typeof(DescriptionAttribute)) {
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has a description attribute.",
                                pInfo[0].Name, mInfo.Name);
                            Console.WriteLine("The description is: \"
{0}\"",

((DescriptionAttribute)attr).Description);
```

```
                        }
                    }
                }
            }
        }
    }
}

/*
 * Output:
 * Parameter args for method ParamArrayAndDesc has the ParamArray
attribute.
 * Parameter args for method ParamArrayAndDesc has a description
attribute.
 * The description is: "This argument is a ParamArray"
 */
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(MemberInfo, Boolean)

Retrieves an array of the custom attributes applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.

C#

```
public static Attribute[] GetCustomAttributes
```

```
(System.Reflection.MemberInfo element, bool inherit);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` is `null`.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.Security;
using System.Runtime.InteropServices;

namespace CustAttrs4CS
{
```

```
    // Define an enumeration of Win32 unmanaged types
    public enum UnmanagedType
    {
        User,
        GDI,
        Kernel,
        Shell,
        Networking,
        Multimedia
    }

    // Define the Unmanaged attribute.
    public class UnmanagedAttribute : Attribute
    {
        // Storage for the UnmanagedType value.
        protected UnmanagedType thisType;

        // Set the unmanaged type in the constructor.
        public UnmanagedAttribute(UnmanagedType type)
        {
            thisType = type;
        }

        // Define a property to get and set the UnmanagedType value.
        public UnmanagedType Win32Type
        {
            get { return thisType; }
            set { thisType = Win32Type; }
        }
    }

    // Create a class for an imported Win32 unmanaged function.
    public class Win32 {
        [DllImport("user32.dll", CharSet = CharSet.Unicode)]
        public static extern int MessageBox(int hWnd, String text,
            String caption, uint type);
    }

    public class AClass {
        // Add some attributes to Win32CallMethod.
        [Obsolete("This method is obsolete. Use managed MsgBox
instead.")]
        [Unmanaged(UnmanagedType.User)]
        public void Win32CallMethod()
        {
            Win32.MessageBox(0, "This is an unmanaged call.", "Caution!",
0);
        }
    }

    class DemoClass {
        static void Main(string[] args)
            {
            // Get the AClass type to access its metadata.
```

```
            Type clsType = typeof(AClass);
            // Get the type information for Win32CallMethod.
            MethodInfo mInfo = clsType.GetMethod("Win32CallMethod");
            if (mInfo != null)
            {
                // Iterate through all the attributes of the method.
                foreach(Attribute attr in
                    Attribute.GetCustomAttributes(mInfo)) {
                    // Check for the Obsolete attribute.
                    if (attr.GetType() == typeof(ObsoleteAttribute))
                    {
                        Console.WriteLine("Method {0} is obsolete. " +
                            "The message is:",
                            mInfo.Name);
                        Console.WriteLine("  \"{0}\"",
                            ((ObsoleteAttribute)attr).Message);
                    }

                    // Check for the Unmanaged attribute.
                    else if (attr.GetType() ==
typeof(UnmanagedAttribute))
                    {
                        Console.WriteLine(
                            "This method calls unmanaged code.");
                        Console.WriteLine(
                            String.Format("The Unmanaged attribute type
is {0}.",

((UnmanagedAttribute)attr).Win32Type));
                        AClass myCls = new AClass();
                        myCls.Win32CallMethod();
                    }
                }
            }
        }
    }
}

/*

This code example produces the following results.

First, the compilation yields the warning, "... This method is
obsolete. Use managed MsgBox instead."
Second, execution yields a message box with a title of "Caution!"
and message text of "This is an unmanaged call."
Third, the following text is displayed in the console window:

Method Win32CallMethod is obsolete. The message is:
  "This method is obsolete. Use managed MsgBox instead."
This method calls unmanaged code.
The Unmanaged attribute type is User.

*/
```

Remarks

The return value contains the custom attributes for ancestors of `element` if `inherit` is `true`.

> ⓘ **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on methods, constructors, and types if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Assembly, Type)

Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.

C#

```
public static Attribute[] GetCustomAttributes (System.Reflection.Assembly
element, Type attributeType);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Attribute[]

An Attribute array that contains the custom attributes of type `attributeType` applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking an Assembly as a parameter.

C#

```
using System;
using System.Reflection;

[assembly: AssemblyTitle("CustAttrs1CS")]
[assembly: AssemblyDescription("GetCustomAttributes() Demo")]
[assembly: AssemblyCompany("Microsoft")]

class Example {
    static void Main() {
        // Get the Assembly object to access its metadata.
        Assembly assy = typeof(Example).Assembly;

        // Iterate through the attributes for the assembly.
        foreach(Attribute attr in Attribute.GetCustomAttributes(assy)) {
            // Check for the AssemblyTitle attribute.
            if (attr.GetType() == typeof(AssemblyTitleAttribute))
                Console.WriteLine("Assembly title is \"{0}\".",
                    ((AssemblyTitleAttribute)attr).Title);
```

```
            // Check for the AssemblyDescription attribute.
            else if (attr.GetType() ==
                typeof(AssemblyDescriptionAttribute))
                Console.WriteLine("Assembly description is \"{0}\".",
                    ((AssemblyDescriptionAttribute)attr).Description);

            // Check for the AssemblyCompany attribute.
            else if (attr.GetType() == typeof(AssemblyCompanyAttribute))
                Console.WriteLine("Assembly company is {0}.",
                    ((AssemblyCompanyAttribute)attr).Company);
        }
    }
}
// The example displays the following output:
//     Assembly title is "CustAttrs1CS".
//     Assembly description is "GetCustomAttributes() Demo".
//     Assembly company is Microsoft.
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Assembly, Boolean)

Retrieves an array of the custom attributes applied to an assembly. Parameters specify the assembly, and an ignored search option.

```C#
public static Attribute[] GetCustomAttributes (System.Reflection.Assembly element, bool inherit);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking an Assembly as a parameter.

```C#
using System;
using System.Reflection;

[assembly: AssemblyTitle("CustAttrs1CS")]
[assembly: AssemblyDescription("GetCustomAttributes() Demo")]
[assembly: AssemblyCompany("Microsoft")]

class Example {
```

```
    static void Main() {
        // Get the Assembly object to access its metadata.
        Assembly assy = typeof(Example).Assembly;

        // Iterate through the attributes for the assembly.
        foreach(Attribute attr in Attribute.GetCustomAttributes(assy)) {
            // Check for the AssemblyTitle attribute.
            if (attr.GetType() == typeof(AssemblyTitleAttribute))
                Console.WriteLine("Assembly title is \"{0}\".",
                    ((AssemblyTitleAttribute)attr).Title);

            // Check for the AssemblyDescription attribute.
            else if (attr.GetType() ==
                typeof(AssemblyDescriptionAttribute))
                Console.WriteLine("Assembly description is \"{0}\".",
                    ((AssemblyDescriptionAttribute)attr).Description);

            // Check for the AssemblyCompany attribute.
            else if (attr.GetType() == typeof(AssemblyCompanyAttribute))
                Console.WriteLine("Assembly company is {0}.",
                    ((AssemblyCompanyAttribute)attr).Company);
        }
    }
}
// The example displays the following output:
//     Assembly title is "CustAttrs1CS".
//     Assembly description is "GetCustomAttributes() Demo".
//     Assembly company is Microsoft.
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(ParameterInfo)

Retrieves an array of the custom attributes applied to a method parameter. A parameter specifies the method parameter.

```
C#

public static Attribute[] GetCustomAttributes
(System.Reflection.ParameterInfo element);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` is `null`.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a ParameterInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

namespace CustAttrs5CS {
    public class AClass {
        public void ParamArrayAndDesc(
            // Add ParamArray (with the keyword) and Description
attributes.
            [Description("This argument is a ParamArray")]
            params int[] args)
        {}
    }

    class DemoClass {
        static void Main(string[] args) {
            // Get the Class type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for the method.
            MethodInfo mInfo = clsType.GetMethod("ParamArrayAndDesc");
            if (mInfo != null) {
                // Get the parameter information.
                ParameterInfo[] pInfo = mInfo.GetParameters();
                if (pInfo != null) {
                    // Iterate through all the attributes for the
parameter.
                    foreach(Attribute attr in
                        Attribute.GetCustomAttributes(pInfo[0])) {
                        // Check for the ParamArray attribute.
                        if (attr.GetType() ==
typeof(ParamArrayAttribute))
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has the ParamArray attribute.",
                                pInfo[0].Name, mInfo.Name);
                        // Check for the Description attribute.
                        else if (attr.GetType() ==
                            typeof(DescriptionAttribute)) {
                            Console.WriteLine("Parameter {0} for method
{1} " +
                                "has a description attribute.",
                                pInfo[0].Name, mInfo.Name);
                            Console.WriteLine("The description is: \"
{0}\"",

((DescriptionAttribute)attr).Description);
                        }
                    }
                }
            }
        }
    }
```

```
}

/*
 * Output:
 * Parameter args for method ParamArrayAndDesc has the ParamArray
attribute.
 * Parameter args for method ParamArrayAndDesc has a description
attribute.
 * The description is: "This argument is a ParamArray"
 */
```

Remarks

If `element` represents a parameter in a method of a derived type, the return value includes the inheritable custom attributes applied to the same parameter in the overridden base methods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Module)

Retrieves an array of the custom attributes applied to a module. A parameter specifies the module.

```C#
public static Attribute[] GetCustomAttributes (System.Reflection.Module
element);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` is `null`.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a Module as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

// Assign some attributes to the module.
[module:Description("A sample description")]

// Set the module's CLSCompliant attribute to false
// The CLSCompliant attribute is applicable for /target:module.
[module:CLSCompliant(false)]

namespace CustAttrs2CS {
    class DemoClass {
        static void Main(string[] args) {
            Type clsType = typeof(DemoClass);
            // Get the Module type to access its metadata.
            Module module = clsType.Module;

            // Iterate through all the attributes for the module.
            foreach(Attribute attr in
Attribute.GetCustomAttributes(module)) {
                // Check for the Description attribute.
                if (attr.GetType() == typeof(DescriptionAttribute))
                    Console.WriteLine("Module {0} has the description " +
                        "\"{1}\".", module.Name,
                        ((DescriptionAttribute)attr).Description);
                // Check for the CLSCompliant attribute.
                else if (attr.GetType() == typeof(CLSCompliantAttribute))
```

```
                        Console.WriteLine("Module {0} {1} CLSCompliant.",
                            module.Name,
                            ((CLSCompliantAttribute)attr).IsCompliant ?
                                "is" : "is not");
                }
            }
        }
    }

    /*
     * Output:
     * Module CustAttrs2CS.exe is not CLSCompliant.
     * Module CustAttrs2CS.exe has the description "A sample description".
     */
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(MemberInfo)

Retrieves an array of the custom attributes applied to a member of a type. A parameter specifies the member.

C#

```
public static Attribute[] GetCustomAttributes
(System.Reflection.MemberInfo element);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, or property member of a class.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` is `null`.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

TypeLoadException

A custom attribute type cannot be loaded.

Examples

The following code example demonstrates the use of GetCustomAttribute, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;
using System.Security;
using System.Runtime.InteropServices;

namespace CustAttrs4CS
{

    // Define an enumeration of Win32 unmanaged types
    public enum UnmanagedType
    {
        User,
        GDI,
        Kernel,
        Shell,
        Networking,
        Multimedia
    }

    // Define the Unmanaged attribute.
    public class UnmanagedAttribute : Attribute
    {
        // Storage for the UnmanagedType value.
```

```
        protected UnmanagedType thisType;

        // Set the unmanaged type in the constructor.
        public UnmanagedAttribute(UnmanagedType type)
        {
            thisType = type;
        }

        // Define a property to get and set the UnmanagedType value.
        public UnmanagedType Win32Type
        {
            get { return thisType; }
            set { thisType = Win32Type; }
        }
    }

    // Create a class for an imported Win32 unmanaged function.
    public class Win32 {
        [DllImport("user32.dll", CharSet = CharSet.Unicode)]
        public static extern int MessageBox(int hWnd, String text,
            String caption, uint type);
    }

    public class AClass {
        // Add some attributes to Win32CallMethod.
        [Obsolete("This method is obsolete. Use managed MsgBox
instead.")]
        [Unmanaged(UnmanagedType.User)]
        public void Win32CallMethod()
        {
            Win32.MessageBox(0, "This is an unmanaged call.", "Caution!",
0);
        }
    }

    class DemoClass {
        static void Main(string[] args)
            {
            // Get the AClass type to access its metadata.
            Type clsType = typeof(AClass);
            // Get the type information for Win32CallMethod.
            MethodInfo mInfo = clsType.GetMethod("Win32CallMethod");
            if (mInfo != null)
            {
                // Iterate through all the attributes of the method.
                foreach(Attribute attr in
                    Attribute.GetCustomAttributes(mInfo)) {
                    // Check for the Obsolete attribute.
                    if (attr.GetType() == typeof(ObsoleteAttribute))
                    {
                        Console.WriteLine("Method {0} is obsolete. " +
                            "The message is:",
                            mInfo.Name);
                        Console.WriteLine("  \"{0}\"",
                            ((ObsoleteAttribute)attr).Message);
```

```
                    }

                    // Check for the Unmanaged attribute.
                    else if (attr.GetType() ==
typeof(UnmanagedAttribute))
                    {
                        Console.WriteLine(
                            "This method calls unmanaged code.");
                        Console.WriteLine(
                            String.Format("The Unmanaged attribute type
is {0}.",

((UnmanagedAttribute)attr).Win32Type));
                        AClass myCls = new AClass();
                        myCls.Win32CallMethod();
                    }
                }
            }
        }
    }
}

/*

This code example produces the following results.

First, the compilation yields the warning, "... This method is
obsolete. Use managed MsgBox instead."
Second, execution yields a message box with a title of "Caution!"
and message text of "This is an unmanaged call."
Third, the following text is displayed in the console window:

Method Win32CallMethod is obsolete. The message is:
  "This method is obsolete. Use managed MsgBox instead."
This method calls unmanaged code.
The Unmanaged attribute type is User.

*/
```

Remarks

The return value contains the custom attributes for ancestors of `element`.

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns security attributes on methods, constructors, and types if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions

of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Assembly)

Retrieves an array of the custom attributes applied to an assembly. A parameter specifies the assembly.

C#

```
public static Attribute[] GetCustomAttributes (System.Reflection.Assembly
element);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` is `null`.

Examples

The following example retrieves the custom attributes found in the current assembly.

C#

```
using System;
using System.Reflection;

[assembly: AssemblyTitle("CustAttrs1CS")]
[assembly: AssemblyDescription("GetCustomAttributes() Demo")]
[assembly: AssemblyCompany("Microsoft")]

class Example {
    static void Main() {
        // Get the Assembly object to access its metadata.
        Assembly assy = typeof(Example).Assembly;

        // Iterate through the attributes for the assembly.
        foreach(Attribute attr in Attribute.GetCustomAttributes(assy)) {
            // Check for the AssemblyTitle attribute.
            if (attr.GetType() == typeof(AssemblyTitleAttribute))
                Console.WriteLine("Assembly title is \"{0}\".",
                    ((AssemblyTitleAttribute)attr).Title);

            // Check for the AssemblyDescription attribute.
            else if (attr.GetType() ==
                typeof(AssemblyDescriptionAttribute))
                Console.WriteLine("Assembly description is \"{0}\".",
                    ((AssemblyDescriptionAttribute)attr).Description);

            // Check for the AssemblyCompany attribute.
            else if (attr.GetType() == typeof(AssemblyCompanyAttribute))
                Console.WriteLine("Assembly company is {0}.",
                    ((AssemblyCompanyAttribute)attr).Company);
        }
    }
}
// The example displays the following output:
//     Assembly title is "CustAttrs1CS".
//     Assembly description is "GetCustomAttributes() Demo".
//     Assembly company is Microsoft.
```

Remarks

ⓘ Note

Starting with the .NET Framework version 2.0, this method returns security attributes if the attributes are stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetCustomAttributes(Module, Boolean)

Retrieves an array of the custom attributes applied to a module. Parameters specify the module, and an ignored search option.

C#

```
public static Attribute[] GetCustomAttributes (System.Reflection.Module
element, bool inherit);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Attribute[]

An Attribute array that contains the custom attributes applied to `element`, or an empty array if no such custom attributes exist.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

Examples

The following code example demonstrates the use of GetCustomAttributes, taking a Module as a parameter.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;

// Assign some attributes to the module.
[module:Description("A sample description")]

// Set the module's CLSCompliant attribute to false
// The CLSCompliant attribute is applicable for /target:module.
[module:CLSCompliant(false)]

namespace CustAttrs2CS {
    class DemoClass {
        static void Main(string[] args) {
            Type clsType = typeof(DemoClass);
            // Get the Module type to access its metadata.
            Module module = clsType.Module;

            // Iterate through all the attributes for the module.
            foreach(Attribute attr in
Attribute.GetCustomAttributes(module)) {
                // Check for the Description attribute.
                if (attr.GetType() == typeof(DescriptionAttribute))
                    Console.WriteLine("Module {0} has the description " +
                        "\"{1}\".", module.Name,
                        ((DescriptionAttribute)attr).Description);
                // Check for the CLSCompliant attribute.
                else if (attr.GetType() == typeof(CLSCompliantAttribute))
                    Console.WriteLine("Module {0} {1} CLSCompliant.",
                        module.Name,
                        ((CLSCompliantAttribute)attr).IsCompliant ?
                            "is" : "is not");
            }
        }
    }
```

```
}

/*
 * Output:
 * Module CustAttrs2CS.exe is not CLSCompliant.
 * Module CustAttrs2CS.exe has the description "A sample description".
 */
```

Remarks

The return value contains the custom attributes for ancestors of `element` if `inherit` is `true`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Examples

The following code example illustrates the use of GetHashCode in the context of Attribute.

C#

```
using System;
using System.Reflection;
using System.Collections.Generic;

// A custom attribute to allow two authors per method.
public class AuthorsAttribute : Attribute
{
   protected string _authorName1;
   protected string _authorName2;

   public AuthorsAttribute(string name1, string name2)
   {
      _authorName1 = name1;
      _authorName2 = name2;
   }

   public string AuthorName1
```

```
    {
       get { return _authorName1; }
       set { _authorName1 = value; }
    }

    public string AuthorName2
    {
       get { return _authorName2; }
       set { _authorName2 = value; }
    }

    // Use the hash code of the string objects and xor them together.
    public override int GetHashCode()
    {
       return _authorName1.GetHashCode() ^ _authorName2.GetHashCode();
    }
}

// Provide the author names for each method of the class.
public class TestClass
{
  [Authors("Immanuel Kant", "Lao Tzu")]
  public void Method1()
  {}

  [Authors("Jean-Paul Sartre", "Friedrich Nietzsche")]
  public void Method2()
  {}

  [Authors("Immanuel Kant", "Lao Tzu")]
  public void Method3()
  {}

  [Authors("Jean-Paul Sartre", "Friedrich Nietzsche")]
  public void Method4()
  {}

  [Authors("Immanuel Kant", "Friedrich Nietzsche")]
  public void Method5()
  {}
}

class Example
{
    static void Main()
    {
       // Get the class type to access its metadata.
       Type clsType = typeof(TestClass);

       // Store author information in a list of tuples.
       var authorsInfo = new List<Tuple<String, AuthorsAttribute>>();

       // Iterate through all the methods of the class.
       foreach(var method in clsType.GetMethods())
       {
```

```
            // Get the Authors attribute for the method if it exists.
            AuthorsAttribute authAttr =
                (AuthorsAttribute)Attribute.GetCustomAttribute(
                method, typeof(AuthorsAttribute));
            if (authAttr != null)
               // Add the information to the author list.
               authorsInfo.Add(Tuple.Create(clsType.Name + "." + method.Name,
                                            authAttr));
        }

        // Iterate through the list
        bool[] listed = new bool[authorsInfo.Count];
        Console.WriteLine("Method authors:\n");

        for (int ctr = 0; ctr < authorsInfo.Count; ctr++) {
           var authorInfo = authorsInfo[ctr];
           if (!listed[ctr]) {
               Console.WriteLine("{0} and {1}", authorInfo.Item2.AuthorName1,
                                                authorInfo.Item2.AuthorName2);
               listed[ctr] = true;
               Console.WriteLine("   {0}", authorInfo.Item1);
               for (int ctr2 = ctr + 1; ctr2 < authorsInfo.Count; ctr2++) {
                   if (!listed[ctr2])
                      if (authorInfo.Item2.Equals(authorsInfo[ctr2].Item2)) {
                         Console.WriteLine("   {0}", authorsInfo[ctr2].Item1);
                         listed[ctr2] = true;
                      }
               }
           }
        }
    }
}
// The example displays the following output:
//       Method authors:
//
//       Immanuel Kant and Lao Tzu
//          TestClass.Method1
//          TestClass.Method3
//       Jean-Paul Sartre and Friedrich Nietzsche
//          TestClass.Method2
//          TestClass.Method4
//       Immanuel Kant and Friedrich Nietzsche
//          TestClass.Method5
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute.IsDefaultAttribute Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class.

C#

```
public virtual bool IsDefaultAttribute ();
```

Returns

Boolean

`true` if this instance is the default attribute for the class; otherwise, `false`.

Examples

The following code example illustrates the use of IsDefaultAttribute.

C#

```
using System;
using System.Reflection;

namespace DefAttrCS
{
    // An enumeration of animals. Start at 1 (0 = uninitialized).
    public enum Animal
    {
        // Pets.
        Dog = 1,
        Cat,
        Bird,
    }

    // A custom attribute to allow a target to have a pet.
    public class AnimalTypeAttribute : Attribute
    {
```

```
        // The constructor is called when the attribute is set.
        public AnimalTypeAttribute(Animal pet)
        {
            thePet = pet;
        }

        // Provide a default constructor and make Dog the default.
        public AnimalTypeAttribute()
        {
            thePet = Animal.Dog;
        }

        // Keep a variable internally ...
        protected Animal thePet;

        // .. and show a copy to the outside world.
        public Animal Pet
        {
            get { return thePet; }
            set { thePet = Pet; }
        }

        // Override IsDefaultAttribute to return the correct response.
        public override bool IsDefaultAttribute()
        {
            if (thePet == Animal.Dog)
                return true;

            return false;
        }
    }

    public class TestClass
    {
        // Use the default constructor.
        [AnimalType]
        public void Method1()
        {}
    }

    class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get type information for the method.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // Get the AnimalType attribute for the method.
            AnimalTypeAttribute atAttr =
                (AnimalTypeAttribute)Attribute.GetCustomAttribute(mInfo,
                typeof(AnimalTypeAttribute));
            // Check to see if the default attribute is applied.
            Console.WriteLine("The attribute {0} for method {1} in class
{2}",
```

```
                atAttr.Pet, mInfo.Name, clsType.Name);
            Console.WriteLine("{0} the default for the AnimalType
attribute.",
                atAttr.IsDefaultAttribute() ? "is" : "is not");
        }
    }
}
```

Remarks

The default implementation of this class returns `false`, and must be implemented in the derived class to be useful to that class.

The implementation of this method in a derived class compares the value of this instance to a standard default value, then returns a Boolean value that indicates whether the value of this instance is equal to the standard value. The standard value is typically coded as a constant in the implementation, or stored programmatically in a field used by the implementation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute.IsDefined Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether any custom attributes of a specified type are applied to an assembly, module, type member, or method parameter.

Overloads

Expand table

IsDefined(ParameterInfo, Type, Boolean)	Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.
IsDefined(Module, Type, Boolean)	Determines whether any custom attributes are applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.
IsDefined(MemberInfo, Type, Boolean)	Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.
IsDefined(Assembly, Type, Boolean)	Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.
IsDefined(MemberInfo, Type)	Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.
IsDefined(Module, Type)	Determines whether any custom attributes of a specified type are applied to a module. Parameters specify the module, and the type of the custom attribute to search for.
IsDefined(Assembly, Type)	Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.

IsDefined(ParameterInfo, Type)	Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.

IsDefined(ParameterInfo, Type, Boolean)

Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, the type of the custom attribute to search for, and whether to search ancestors of the method parameter.

```
C#

public static bool IsDefined (System.Reflection.ParameterInfo element,
Type attributeType, bool inherit);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

ExecutionEngineException

`element` is not a method, constructor, or type.

Examples

The following code example illustrates the use of IsDefined, taking a ParameterInfo as a parameter.

C#

```
using System;
using System.Reflection;

namespace IsDef5CS
{
    public class TestClass
    {
        // Assign a ParamArray attribute to the parameter using the
keyword.
        public void Method1(params String[] args)
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // Get the ParameterInfo array for the method parameters.
            ParameterInfo[] pInfo = mInfo.GetParameters();
            if (pInfo != null)
            {
                // See if the ParamArray attribute is defined.
                bool isDef = Attribute.IsDefined(pInfo[0],

typeof(ParamArrayAttribute));
                // Display the result.
                Console.WriteLine("The ParamArray attribute {0} defined
for " +
                                  "parameter {1} of method {2}.",
                                  isDef ? "is" : "is not",
                                  pInfo[0].Name,
                                  mInfo.Name);
            }
            else
                Console.WriteLine("The parameters information could " +
```

```
                                "not be retrieved for method {0}.",
mInfo.Name);
        }
    }
}

/*
 * Output:
 * The ParamArray attribute is defined for parameter args of method
Method1.
 */
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(Module, Type, Boolean)

Determines whether any custom attributes are applied to a module. Parameters specify the module, the type of the custom attribute to search for, and an ignored search option.

C#

```
public static bool IsDefined (System.Reflection.Module element, Type
attributeType, bool inherit);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example illustrates the use of IsDefined, taking a Module as a parameter.

C#

```
using System;
using System.Diagnostics;

// Add the Debuggable attribute to the module.
[module:Debuggable(true, false)]
namespace IsDef2CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // See if the Debuggable attribute is defined for this
module.
            bool isDef = Attribute.IsDefined(clsType.Module,
                typeof(DebuggableAttribute));
            // Display the result.
            Console.WriteLine("The Debuggable attribute {0} " +
                "defined for Module {1}.",
                isDef ? "is" : "is not",
                clsType.Module.Name);
```

```
            // If the attribute is defined, display the JIT settings.
            if (isDef)
            {
                // Retrieve the attribute itself.
                DebuggableAttribute dbgAttr = (DebuggableAttribute)
                    Attribute.GetCustomAttribute(clsType.Module,
                    typeof(DebuggableAttribute));
                if (dbgAttr != null)
                {
                    Console.WriteLine("JITTrackingEnabled is {0}.",
                        dbgAttr.IsJITTrackingEnabled);
                    Console.WriteLine("JITOptimizerDisabled is {0}.",
                        dbgAttr.IsJITOptimizerDisabled);
                }
                else
                    Console.WriteLine("The Debuggable attribute " +
                        "could not be retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Debuggable attribute is defined for Module IsDef2CS.exe.
 * JITTrackingEnabled is True.
 * JITOptimizerDisabled is False.
 */
```

Remarks

This method ignores the `inherit` parameter and does not search the ancestors of `element` for custom attributes.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(MemberInfo, Type, Boolean)

Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, the type of the custom attribute to search for, and whether to search ancestors of the member.

```C#
public static bool IsDefined (System.Reflection.MemberInfo element, Type attributeType, bool inherit);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, type, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

If `true`, specifies to also search the ancestors of `element` for custom attributes.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

Examples

The following code example illustrates the use of IsDefined, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;

namespace IsDef4CS
{
    public class TestClass
    {
        // Assign the Obsolete attribute to a method.
        [Obsolete("This method is obsolete. Use Method2 instead.")]
        public void Method1()
        {}
        public void Method2()
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // See if the Obsolete attribute is defined for this method.
            bool isDef = Attribute.IsDefined(mInfo,
typeof(ObsoleteAttribute));
            // Display the result.
            Console.WriteLine("The Obsolete Attribute {0} defined for {1}
of class {2}.",
                isDef ? "is" : "is not", mInfo.Name, clsType.Name);
            // If it's defined, display the attribute's message.
            if (isDef)
            {
                ObsoleteAttribute obsAttr =

(ObsoleteAttribute)Attribute.GetCustomAttribute(
                                                        mInfo,
typeof(ObsoleteAttribute));
                if (obsAttr != null)
                    Console.WriteLine("The message is: \"{0}\".",
                        obsAttr.Message);
                else
                    Console.WriteLine("The message could not be
retrieved.");
            }
        }
    }
}
```

```
/*
 * Output:
 * The Obsolete Attribute is defined for Method1 of class TestClass.
 * The message is: "This method is obsolete. Use Method2 instead.".
 */
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns `true` if a type, method, or constructor has security attributes stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(Assembly, Type, Boolean)

Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, the type of the custom attribute to search for, and an ignored search option.

C#

```
public static bool IsDefined (System.Reflection.Assembly element, Type
attributeType, bool inherit);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

`inherit` Boolean

This parameter is ignored, and does not affect the operation of this method.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example illustrates the use of IsDefined, taking an Assembly as a parameter.

C#

```
using System;
using System.Reflection;

// Add an AssemblyDescription attribute
[assembly: AssemblyDescription("A sample description")]
namespace IsDef1CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
```

```
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // Get the assembly object.
            Assembly assy = clsType.Assembly;
            // Store the assembly's name.
            String assyName = assy.GetName().Name;
            // See if the Assembly Description is defined.
            bool isdef = Attribute.IsDefined(assy,
                typeof(AssemblyDescriptionAttribute));
            if (isdef)
            {
                // Affirm that the attribute is defined.
                Console.WriteLine("The AssemblyDescription attribute " +
                    "is defined for assembly {0}.", assyName);
                // Get the description attribute itself.
                AssemblyDescriptionAttribute adAttr =

(AssemblyDescriptionAttribute)Attribute.GetCustomAttribute(
                    assy, typeof(AssemblyDescriptionAttribute));
                // Display the description.
                if (adAttr != null)
                    Console.WriteLine("The description is \"{0}\".",
                        adAttr.Description);
                else
                    Console.WriteLine("The description could not " +
                        "be retrieved.");
            }
            else
                Console.WriteLine("The AssemblyDescription attribute is
not " +
                    "defined for assembly {0}.", assyName);
        }
    }
}

/*
 * Output:
 * The AssemblyDescription attribute is defined for assembly IsDef1CS.
 * The description is "A sample description".
 */
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns `true` if the assembly has security attributes stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and

assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(MemberInfo, Type)

Determines whether any custom attributes are applied to a member of a type. Parameters specify the member, and the type of the custom attribute to search for.

C#

```
public static bool IsDefined (System.Reflection.MemberInfo element, Type
attributeType);
```

Parameters

`element` MemberInfo

An object derived from the MemberInfo class that describes a constructor, event, field, method, type, or property member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

NotSupportedException

`element` is not a constructor, method, property, event, type, or field.

Examples

The following code example illustrates the use of IsDefined, taking a MemberInfo as a parameter.

C#

```
using System;
using System.Reflection;

namespace IsDef4CS
{
    public class TestClass
    {
        // Assign the Obsolete attribute to a method.
        [Obsolete("This method is obsolete. Use Method2 instead.")]
        public void Method1()
        {}
        public void Method2()
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // See if the Obsolete attribute is defined for this method.
            bool isDef = Attribute.IsDefined(mInfo,
typeof(ObsoleteAttribute));
            // Display the result.
            Console.WriteLine("The Obsolete Attribute {0} defined for {1}
of class {2}.",
                isDef ? "is" : "is not", mInfo.Name, clsType.Name);
            // If it's defined, display the attribute's message.
            if (isDef)
```

```
            {
                ObsoleteAttribute obsAttr =

(ObsoleteAttribute)Attribute.GetCustomAttribute(
                                                        mInfo,
typeof(ObsoleteAttribute));
                if (obsAttr != null)
                    Console.WriteLine("The message is: \"{0}\".",
                        obsAttr.Message);
                else
                    Console.WriteLine("The message could not be
retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Obsolete Attribute is defined for Method1 of class TestClass.
 * The message is: "This method is obsolete. Use Method2 instead.".
 */
```

Remarks

The ancestors of `element` are searched for custom attributes.

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns `true` if a type, method, or constructor has security attributes stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

IsDefined(Module, Type)

Determines whether any custom attributes of a specified type are applied to a module. Parameters specify the module, and the type of the custom attribute to search for.

C#

```
public static bool IsDefined (System.Reflection.Module element, Type
attributeType);
```

Parameters

`element` Module

An object derived from the Module class that describes a portable executable file.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example illustrates the use of IsDefined, taking a Module as a parameter.

C#

```
using System;
using System.Diagnostics;

// Add the Debuggable attribute to the module.
[module:Debuggable(true, false)]
namespace IsDef2CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // See if the Debuggable attribute is defined for this
module.
            bool isDef = Attribute.IsDefined(clsType.Module,
                typeof(DebuggableAttribute));
            // Display the result.
            Console.WriteLine("The Debuggable attribute {0} " +
                "defined for Module {1}.",
                isDef ? "is" : "is not",
                clsType.Module.Name);
            // If the attribute is defined, display the JIT settings.
            if (isDef)
            {
                // Retrieve the attribute itself.
                DebuggableAttribute dbgAttr = (DebuggableAttribute)
                    Attribute.GetCustomAttribute(clsType.Module,
                    typeof(DebuggableAttribute));
                if (dbgAttr != null)
                {
                    Console.WriteLine("JITTrackingEnabled is {0}.",
                        dbgAttr.IsJITTrackingEnabled);
                    Console.WriteLine("JITOptimizerDisabled is {0}.",
                        dbgAttr.IsJITOptimizerDisabled);
                }
                else
                    Console.WriteLine("The Debuggable attribute " +
                        "could not be retrieved.");
            }
        }
    }
}

/*
 * Output:
 * The Debuggable attribute is defined for Module IsDef2CS.exe.
 * JITTrackingEnabled is True.
```

```
  * JITOptimizerDisabled is False.
  */
```

Remarks

The ancestors of `element` are not searched for custom attributes.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(Assembly, Type)

Determines whether any custom attributes are applied to an assembly. Parameters specify the assembly, and the type of the custom attribute to search for.

C#

```
public static bool IsDefined (System.Reflection.Assembly element, Type
attributeType);
```

Parameters

`element` Assembly

An object derived from the Assembly class that describes a reusable collection of modules.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example illustrates the use of IsDefined, taking an Assembly as a parameter.

C#

```
using System;
using System.Reflection;

// Add an AssemblyDescription attribute
[assembly: AssemblyDescription("A sample description")]
namespace IsDef1CS
{
    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(DemoClass);
            // Get the assembly object.
            Assembly assy = clsType.Assembly;
            // Store the assembly's name.
            String assyName = assy.GetName().Name;
            // See if the Assembly Description is defined.
            bool isdef = Attribute.IsDefined(assy,
                typeof(AssemblyDescriptionAttribute));
            if (isdef)
            {
                // Affirm that the attribute is defined.
                Console.WriteLine("The AssemblyDescription attribute " +
                    "is defined for assembly {0}.", assyName);
                // Get the description attribute itself.
                AssemblyDescriptionAttribute adAttr =

(AssemblyDescriptionAttribute)Attribute.GetCustomAttribute(
                    assy, typeof(AssemblyDescriptionAttribute));
                // Display the description.
```

```
                if (adAttr != null)
                    Console.WriteLine("The description is \"{0}\".",
                        adAttr.Description);
                else
                    Console.WriteLine("The description could not " +
                        "be retrieved.");
            }
            else
                Console.WriteLine("The AssemblyDescription attribute is
not " +
                    "defined for assembly {0}.", assyName);
        }
    }
}

/*
 * Output:
 * The AssemblyDescription attribute is defined for assembly IsDef1CS.
 * The description is "A sample description".
 */
```

Remarks

> **Note**
>
> Starting with the .NET Framework version 2.0, this method returns `true` if the assembly has security attributes stored in the new metadata format. Assemblies compiled with version 2.0 or later use the new format. Dynamic assemblies and assemblies compiled with earlier versions of the .NET Framework use the old XML format. See **Emitting Declarative Security Attributes**.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDefined(ParameterInfo, Type)

Determines whether any custom attributes are applied to a method parameter. Parameters specify the method parameter, and the type of the custom attribute to search for.

```
C#

public static bool IsDefined (System.Reflection.ParameterInfo element,
Type attributeType);
```

Parameters

`element` ParameterInfo

An object derived from the ParameterInfo class that describes a parameter of a member of a class.

`attributeType` Type

The type, or a base type, of the custom attribute to search for.

Returns

Boolean

`true` if a custom attribute of type `attributeType` is applied to `element`; otherwise, `false`.

Exceptions

ArgumentNullException

`element` or `attributeType` is `null`.

ArgumentException

`attributeType` is not derived from Attribute.

Examples

The following code example illustrates the use of IsDefined, taking a ParameterInfo as a parameter.

```
C#

using System;
using System.Reflection;
```

```
namespace IsDef5CS
{
    public class TestClass
    {
        // Assign a ParamArray attribute to the parameter using the
keyword.
        public void Method1(params String[] args)
        {}
    }

    public class DemoClass
    {
        static void Main(string[] args)
        {
            // Get the class type to access its metadata.
            Type clsType = typeof(TestClass);
            // Get the MethodInfo object for Method1.
            MethodInfo mInfo = clsType.GetMethod("Method1");
            // Get the ParameterInfo array for the method parameters.
            ParameterInfo[] pInfo = mInfo.GetParameters();
            if (pInfo != null)
            {
                // See if the ParamArray attribute is defined.
                bool isDef = Attribute.IsDefined(pInfo[0],

typeof(ParamArrayAttribute));
                // Display the result.
                Console.WriteLine("The ParamArray attribute {0} defined
for " +
                                  "parameter {1} of method {2}.",
                                  isDef ? "is" : "is not",
                                  pInfo[0].Name,
                                  mInfo.Name);
            }
            else
                Console.WriteLine("The parameters information could " +
                            "not be retrieved for method {0}.",
mInfo.Name);
        }
    }
}

/*
 * Output:
 * The ParamArray attribute is defined for parameter args of method
Method1.
 */
```

Remarks

The ancestors of `element` are searched for custom attributes.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Attribute.Match(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns a value that indicates whether this instance equals a specified object.

```
C#
public virtual bool Match (object? obj);
```

Parameters

obj Object

An Object to compare with this instance of Attribute.

Returns

Boolean

`true` if this instance equals `obj`; otherwise, `false`.

Remarks

This method determines if one Attribute equals another. Its default implementation is the same as Equals, which determines whether two attributes are of the same type and have the same field values.

In general, the Equals method is intended to perform a standard test for reference or value equality. You can override the Match method when you want to perform a custom comparison for equality that is based on some criteria other than that two attribute instances are of the same type and have identical values. For example, you can override the Match method to compare attributes in the following scenarios:

- An attribute includes a bit field, and you consider two attribute instances equal if a particular bit is set. For example, a `NumericDisplay` attribute might include a bit

field that indicates what numeric formats (such as binary, octal, decimal, and hexadecimal) a client supports. The overridden Match method might consider two instances equal if they support the same numeric format.

- An attribute includes multiple fields that contain the same kind of information, or it contains an array in which values can be in any order. For example, an `Author` attribute might include multiple fields for author names. The overridden Match method might consider two instances equal if they have the same authors regardless of whether each field is equal to the corresponding field.

Example

The following example illustrates the use of Match to create a custom comparison method for attribute values. If defines an `AuthorsAttribute` that internally contains a `List<String>` that stores authors' names. Because the names can occur in any order in the list, it overrides the Match method to compare author names regardless of their position in the list. Note the Equals method, which performs a test for value equality, returns different results from the Match method.

C#

```
using System;
using System.Collections.Generic;
using System.Reflection;

// A custom attribute to allow multiple authors per method.
[AttributeUsage(AttributeTargets.Method)]
public class AuthorsAttribute : Attribute
{
   protected List<string> _authors;

    public AuthorsAttribute(params string[] names)
   {
      _authors = new List<string>(names);
    }

   public List<string> Authors
   {
       get { return _authors; }
   }

    // Determine if the object is a match to this one.
    public override bool Match(object obj)
   {
      // Return false if obj is null or not an AuthorsAttribute.
      AuthorsAttribute authors2 = obj as AuthorsAttribute;
      if (authors2 == null) return false;
```

```
        // Return true if obj and this instance are the same object
reference.
      if (Object.ReferenceEquals(this, authors2))
         return true;

      // Return false if obj and this instance have different numbers of
authors
      if (_authors.Count != authors2._authors.Count)
         return false;

      bool matches = false;
      foreach (var author in _authors)
      {
         for (int ctr = 0; ctr < authors2._authors.Count; ctr++)
         {
            if (author == authors2._authors[ctr])
            {
               matches = true;
               break;
            }
            if (ctr == authors2._authors.Count)
            {
               matches = false;
            }
         }
      }
      return matches;
   }

   public override string ToString()
   {
      string retval = "";
      for (int ctr = 0; ctr < _authors.Count; ctr++)
      {
         retval += $"{_authors[ctr]}{(ctr < _authors.Count - 1 ? ", " :
String.Empty)}";
      }
      if (retval.Trim().Length == 0)
      {
         return "<unknown>";
      }
      return retval;
   }
}

// Add some authors to methods of a class.
public class TestClass {
    [Authors("Leo Tolstoy", "John Milton")]
    public void Method1()
    {}

    [Authors("Anonymous")]
    public void Method2()
    {}
```

```
    [Authors("Leo Tolstoy", "John Milton", "Nathaniel Hawthorne")]
    public void Method3()
    {}

    [Authors("John Milton", "Leo Tolstoy")]
    public void Method4()
    {}
}

class Example
{
    static void Main()
   {
        // Get the type for TestClass to access its metadata.
        Type clsType = typeof(TestClass);

        // Iterate through each method of the class.
      AuthorsAttribute authors = null;
        foreach(var method in clsType.GetMethods())
      {
            // Check each method for the Authors attribute.
            AuthorsAttribute authAttr = (AuthorsAttribute)
Attribute.GetCustomAttribute(method,
                                          typeof(AuthorsAttribute));
            if (authAttr != null)
         {
            // Display the authors.
                Console.WriteLine($"{clsType.Name}.{method.Name} was
authored by {authAttr}.");

            // Select Method1's authors as the basis for comparison.
            if (method.Name == "Method1")
            {
                   authors = authAttr;
               Console.WriteLine();
               continue;
            }

            // Compare first authors with the authors of this method.
            if (authors.Match(authAttr))
            {
                    Console.WriteLine("TestClass.Method1 was also authored
by the same team.");
            }

            // Perform an equality comparison of the two attributes.
            Console.WriteLine($"{authors} {(authors.Equals(authAttr) ? "=" :
"<>")} {authAttr}");
            Console.WriteLine();
            }
        }
    }
}
// The example displays the following output:
//       TestClass.Method1 was authored by Leo Tolstoy, John Milton.
```

```
//
//       TestClass.Method2 was authored by Anonymous.
//       Leo Tolstoy, John Milton <> Anonymous
//
//       TestClass.Method3 was authored by Leo Tolstoy, John Milton,
Nathaniel Hawthorne.
//       Leo Tolstoy, John Milton <> Leo Tolstoy, John Milton, Nathaniel
Hawthorne
//
//       TestClass.Method4 was authored by John Milton, Leo Tolstoy.
//       TestClass.Method1 was also authored by the same team.
//       Leo Tolstoy, John Milton <> John Milton, Leo Tolstoy
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AttributeTargets Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the application elements on which it is valid to apply an attribute.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum AttributeTargets
```

Inheritance Object → ValueType → Enum → AttributeTargets

Attributes FlagsAttribute

Fields

Expand table

All	32767	Attribute can be applied to any application element.
Assembly	1	Attribute can be applied to an assembly.
Class	4	Attribute can be applied to a class.
Constructor	32	Attribute can be applied to a constructor.
Delegate	4096	Attribute can be applied to a delegate.
Enum	16	Attribute can be applied to an enumeration.
Event	512	Attribute can be applied to an event.
Field	256	Attribute can be applied to a field.
GenericParameter	16384	Attribute can be applied to a generic parameter. Currently, this attribute can be applied only in C#, Microsoft intermediate language

		(MSIL), and emitted code.
Interface	1024	Attribute can be applied to an interface.
Method	64	Attribute can be applied to a method.
Module	2	Attribute can be applied to a module. `Module` refers to a portable executable file (.dll or.exe) and not a Visual Basic standard module.
Parameter	2048	Attribute can be applied to a parameter.
Property	128	Attribute can be applied to a property.
ReturnValue	8192	Attribute can be applied to a return value.
Struct	8	Attribute can be applied to a structure; that is, a value type.

Examples

The following example illustrates the application of attributes to various targets.

> **Note**
>
> Visual Basic and Visual C++ syntax currently do not support the application of attributes to type parameters.

C#

```
using System;

namespace AttTargsCS {
    // This attribute is only valid on a class.
    [AttributeUsage(AttributeTargets.Class)]
    public class ClassTargetAttribute : Attribute {
    }

    // This attribute is only valid on a method.
    [AttributeUsage(AttributeTargets.Method)]
    public class MethodTargetAttribute : Attribute {
    }

    // This attribute is only valid on a constructor.
    [AttributeUsage(AttributeTargets.Constructor)]
    public class ConstructorTargetAttribute : Attribute {
    }

    // This attribute is only valid on a field.
    [AttributeUsage(AttributeTargets.Field)]
    public class FieldTargetAttribute : Attribute {
```

```
    }

    // This attribute is valid on a class or a method.
    [AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]
    public class ClassMethodTargetAttribute : Attribute {
    }

    // This attribute is valid on a generic type parameter.
    [AttributeUsage(AttributeTargets.GenericParameter)]
    public class GenericParameterTargetAttribute : Attribute {
    }

    // This attribute is valid on any target.
    [AttributeUsage(AttributeTargets.All)]
    public class AllTargetsAttribute : Attribute {
    }

    [ClassTarget]
    [ClassMethodTarget]
    [AllTargets]
    public class TestClassAttribute {
        [ConstructorTarget]
        [AllTargets]
        TestClassAttribute() {
        }

        [MethodTarget]
        [ClassMethodTarget]
        [AllTargets]
        public void Method1() {
        }

        [FieldTarget]
        [AllTargets]
        public int myInt;

        public void GenericMethod<
            [GenericParameterTarget, AllTargets] T>(T x) {
        }

        static void Main(string[] args) {
        }
    }
}
```

Remarks

The AttributeUsageAttribute class uses this enumeration to specify the kind of element on which it is valid to apply an attribute.

AttributeTargets enumeration values can be combined with a bitwise OR operation to get the preferred combination.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- AttributeUsageAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AttributeUsageAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the usage of another attribute class. This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.Class, Inherited=true)]
public sealed class AttributeUsageAttribute : Attribute
```

Inheritance Object → Attribute → AttributeUsageAttribute

Attributes AttributeUsageAttribute

Remarks

When you are defining your own attribute class, you can control the manner in which it is used by placing an AttributeUsageAttribute on your attribute class. The indicated attribute class must derive from Attribute, either directly or indirectly.

Attribute classes have positional and named parameters. Each public constructor for an attribute class defines a valid sequence of positional parameters for that class. Named parameters are defined by the non-static, public, and read-write fields or properties of the attribute class.

The three properties of AttributeUsageAttribute are set by defining the following parameters:

- ValidOn

This positional parameter specifies the program elements that the indicated attribute can be placed on. The set of all possible elements that you can place an attribute on is listed in the AttributeTargets enumeration. You can combine several AttributeTargets values using a bitwise OR operation to get the desired combination of valid program elements.

- AllowMultiple

This named parameter specifies whether the indicated attribute can be specified more than once for a given program element.

- Inherited

This named parameter specifies whether the indicated attribute can be inherited by derived classes and overriding members.

For more information about using attributes, see Attribute and Attributes.

Constructors

Expand table

AttributeUsageAttribute(AttributeTargets)	Initializes a new instance of the AttributeUsageAttribute class with the specified list of AttributeTargets, the AllowMultiple value, and the Inherited value.

Properties

Expand table

AllowMultiple	Gets or sets a Boolean value indicating whether more than one instance of the indicated attribute can be specified for a single program element.
Inherited	Gets or sets a Boolean value that determines whether the indicated attribute is inherited by derived classes and overriding members.
TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)
ValidOn	Gets a set of values identifying which program elements that the indicated attribute can be applied to.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)

GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute
- AttributeTargets
- Extending Metadata Using Attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

AttributeUsage Attribute(AttributeTargets) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the AttributeUsageAttribute class with the specified list of AttributeTargets, the AllowMultiple value, and the Inherited value.

C#

```
public AttributeUsageAttribute (AttributeTargets validOn);
```

Parameters

validOn AttributeTargets

The set of values combined using a bitwise OR operation to indicate which program elements are valid.

Examples

The definition of the DispId attribute illustrates the use of a bitwise OR operation to combine several AttributeTargets values.

C#

```
namespace System.Runtime.InteropServices
{
   [AttributeUsage(AttributeTargets.Method | AttributeTargets.Field |
    AttributeTargets.Property)]
   public class DispIdAttribute: System.Attribute
   {
      public DispIdAttribute(int value) {
        // . . .
      }

      public int Value {
         get {
            // . . .
```

```
            return 0;
        }
    }
  }
}
```

Remarks

You can combine several AttributeTargets values using a bitwise OR operation to get the desired combination of valid program elements.

For default property values, see the ValidOn, AllowMultiple, and Inherited properties.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- AttributeTargets

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AttributeUsageAttribute.AllowMultiple Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets a Boolean value indicating whether more than one instance of the indicated attribute can be specified for a single program element.

```
C#
public bool AllowMultiple { get; set; }
```

Property Value

Boolean

`true` if more than one instance is allowed to be specified; otherwise, `false`. The default is `false`.

Remarks

An attribute that can be specified more than once for a program element is called a multi-use attribute. An attribute that can be specified only once is called a single-use attribute.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AttributeUsageAttribute.Inherited Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets a Boolean value that determines whether the indicated attribute is inherited by derived classes and overriding members.

```
C#
public bool Inherited { get; set; }
```

Property Value

Boolean

`true` if the attribute can be inherited by derived classes and overriding members; otherwise, `false`. The default is `true`.

Examples

The following example illustrates the difference between an attribute to which an AttributeUsageAttribute attribute with an Inherited property value of `true` is applied and one to which AttributeUsageAttribute attribute with an Inherited property value of `false` is applied. The example defines two attributes, `InheritedAttribute` and `NotInheritedAttribute`. Both attributes can apply to classes and methods. Because the Inherited property of the AttributeUsageAttribute attribute applied to `InheritedAttribute` is `true`, it is inherited by derived classes and the members of derived classes that override the base class method. On the other hand, because the Inherited property of the AttributeUsageAttribute attribute applied to `NotInheritedAttribute` is `false`, it is not inherited by derived classes and the members of derived classes that override the base class method.

```
C#
```

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method |
                AttributeTargets.Property | AttributeTargets.Field,
                Inherited = true)]
public class InheritedAttribute : Attribute
{ }

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method |
                AttributeTargets.Property | AttributeTargets.Field,
                Inherited = false)]
public class NotInheritedAttribute : Attribute
{ }
```

The example then defines two base classes. The first, `BaseA`, has a single method, `MethodA`. The second, `BaseB`, has a single method, `MethodB`. `BaseA` and `MethodA` are tagged with the `InheritedAttribute` attribute, and `BaseB` and `MethodB` are tagged with the `NotInheritedAttribute` attribute. `DerivedA` inherits from `BaseA` and overrides its `MethodA` method. `DerivedB` inherits from `BaseB` and overrides its `MethodB` method.

C#

```
using System;
using System.Reflection;

[InheritedAttribute]
public class BaseA
{
    [InheritedAttribute]
    public virtual void MethodA()
    { }
}

public class DerivedA : BaseA
{
    public override void MethodA()
    { }
}

[NotInheritedAttribute]
public class BaseB
{
    [NotInheritedAttribute]
    public virtual void MethodB()
    { }
}

public class DerivedB : BaseB
{
    public override void MethodB()
    { }
}
```

```
public class Example
{
    public static void Main()
    {
        Type typeA = typeof(DerivedA);
        Console.WriteLine($"DerivedA has Inherited attribute:
{typeA.GetCustomAttributes(typeof(InheritedAttribute), true).Length > 0}");
        MethodInfo memberA = typeA.GetMethod(nameof(DerivedA.MethodA));
        Console.WriteLine($"DerivedA.MemberA has Inherited attribute:
{memberA.GetCustomAttributes(typeof(InheritedAttribute), true).Length >
0}\n");

        Type typeB = typeof(DerivedB);
        Console.WriteLine($"DerivedB has NotInherited attribute:
{typeB.GetCustomAttributes(typeof(NotInheritedAttribute), true).Length >
0}");
        MethodInfo memberB = typeB.GetMethod(nameof(DerivedB.MethodB));
        Console.WriteLine($"DerivedB.MemberB has NotInherited attribute:
{memberB.GetCustomAttributes(typeof(NotInheritedAttribute), true).Length >
0}");
    }
}
// The example displays the following output:
//       DerivedA has Inherited attribute: True
//       DerivedA.MemberA has Inherited attribute: True
//
//       DerivedB has NotInherited attribute: False
//       DerivedB.MemberB has NotInherited attribute: False
```

As the output from the example shows, `DerivedA` and `DerivedA.MethodA` inherit the `InheritedAttribute` attribute, but `DerivedB` and `DerivedB.MethodB` do not inherit the `NotInheritedAttribute` attribute.

Remarks

The Inherited property determines:

- Whether classes derived from a base class tagged with the attribute to which the AttributeUsageAttribute attribute is applied inherit that attribute.
- Whether methods of derived classes that override a base class method tagged with the attribute to which the AttributeUsageAttribute attribute is applied inherit that attribute. (If a class inherits a base class member, it also inherits any attributes applied to that member.)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AttributeUsageAttribute.ValidOn Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a set of values identifying which program elements that the indicated attribute can be applied to.

```
C#
public AttributeTargets ValidOn { get; }
```

Property Value

AttributeTargets

One or several AttributeTargets values. The default is `All`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- AttributeTargets

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BadImageFormatException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the file image of a dynamic link library (DLL) or an executable program is invalid.

C#

```
public class BadImageFormatException : SystemException
```

Inheritance Object → Exception → SystemException → BadImageFormatException

Remarks

This exception is thrown when the file format of a dynamic link library (.dll file) or an executable (.exe file) doesn't conform to the format that the common language runtime expects. In particular, the exception is thrown under the following conditions:

- An earlier version of a .NET utility, such as ILDasm.exe or installutil.exe, is used with an assembly that was developed with a later version of .NET.

  To address this exception, use the version of the tool that corresponds to the version of .NET that was used to develop the assembly. This may require modifying the `Path` environment variable or providing a fully qualified path to the correct executable.

- You are trying to load an unmanaged dynamic link library or executable (such as a Windows system DLL) as if it were a .NET assembly. The following example illustrates this by using the Assembly.LoadFile method to load Kernel32.dll.

  C#

  ```
  // Windows DLL (non-.NET assembly)
  string filePath = Environment.ExpandEnvironmentVariables("%windir%");
  if (! filePath.Trim().EndsWith(@"\"))
     filePath += @"\";
  filePath += @"System32\Kernel32.dll";
  ```

```
try {
   Assembly assem = Assembly.LoadFile(filePath);
}
catch (BadImageFormatException e) {
   Console.WriteLine("Unable to load {0}.", filePath);
   Console.WriteLine(e.Message.Substring(0,
                     e.Message.IndexOf(".") + 1));
}
// The example displays an error message like the following:
//       Unable to load C:\WINDOWS\System32\Kernel32.dll.
//       The module was expected to contain an assembly manifest.
```

To address this exception, access the methods defined in the DLL by using the features provided by your development language, such as the `Declare` statement in Visual Basic or the DllImportAttribute attribute with the `extern` keyword in C# and F#.

- You are trying to load a reference assembly in a context other than the reflection-only context. You can address this issue in either of two ways:
  - You can load the implementation assembly rather than the reference assembly.
  - You can load the reference assembly in the reflection-only context by calling the Assembly.ReflectionOnlyLoad method.

- A DLL or executable is loaded as a 64-bit assembly, but it contains 32-bit features or resources. For example, it relies on COM interop or calls methods in a 32-bit dynamic link library.

  To address this exception, set the project's **Platform target** property to x86 (instead of x64 or AnyCPU) and recompile.

- Your application's components were created using different versions of .NET. Typically, this exception occurs when an application or component that was developed using the .NET Framework 1.0 or the .NET Framework 1.1 tries to load an assembly that was developed using the .NET Framework 2.0 SP1 or later, or when an application that was developed using the .NET Framework 2.0 SP1 or .NET Framework 3.5 tries to load an assembly that was developed using the .NET Framework 4 or later. The BadImageFormatException may be reported as a compile-time error, or the exception may be thrown at run time. The following example defines a `StringLib` class that has a single member, `ToProperCase`, and that resides in an assembly named StringLib.dll.

```
C#
```

```
using System;

public class StringLib
{
   private string[] exceptionList = { "a", "an", "the", "in", "on",
"of" };
   private char[] separators = { ' ' };

   public string ToProperCase(string title)
   {
      bool isException = false;

      string[] words = title.Split( separators,
StringSplitOptions.RemoveEmptyEntries);
      string[] newWords = new string[words.Length];

      for (int ctr = 0; ctr <= words.Length - 1; ctr++)
      {
         isException = false;

         foreach (string exception in exceptionList)
         {
            if (words[ctr].Equals(exception) && ctr > 0)
            {
               isException = true;
               break;
            }
         }

         if (! isException)
            newWords[ctr] = words[ctr].Substring(0, 1).ToUpper() +
words[ctr].Substring(1);
         else
            newWords[ctr] = words[ctr];
      }
      return string.Join(" ", newWords);
   }
}
// Attempting to load the StringLib.dll assembly produces the following
output:
//    Unhandled Exception: System.BadImageFormatException:
//                         The format of the file 'StringLib.dll' is
invalid.
```

The following example uses reflection to load an assembly named StringLib.dll. If the source code is compiled with a .NET Framework 1.1 compiler, a BadImageFormatException is thrown by the Assembly.LoadFrom method.

C#

```
using System;
using System.Reflection;
```

```
public class Example
{
   public static void Main()
   {
      string title = "a tale of two cities";
//      object[] args = { title}
      // Load assembly containing StateInfo type.
      Assembly assem = Assembly.LoadFrom(@".\StringLib.dll");
      // Get type representing StateInfo class.
      Type stateInfoType = assem.GetType("StringLib");
      // Get Display method.
      MethodInfo mi = stateInfoType.GetMethod("ToProperCase");
      // Call the Display method.
      string properTitle = (string) mi.Invoke(null, new object[] {
title } );
      Console.WriteLine(properTitle);
   }
}
```

To address this exception, make sure that the assembly whose code is executing and that throws the exception, and the assembly to be loaded, both target compatible versions of .NET.

- The components of your application target different platforms. For example, you are trying to load ARM assemblies in an x86 application. You can use the following command-line utility to determine the target platforms of individual .NET assemblies. The list of files should be provided as a space-delimited list at the command line.

C#

```
using System;
using System.IO;
using System.Reflection;

public class Example
{
   public static void Main()
   {
      String[] args = Environment.GetCommandLineArgs();
      if (args.Length == 1) {
         Console.WriteLine("\nSyntax:   PlatformInfo <filename>\n");
         return;
      }
      Console.WriteLine();

      // Loop through files and display information about their
platform.
      for (int ctr = 1; ctr < args.Length; ctr++) {
         string fn = args[ctr];
```

```
            if (! File.Exists(fn)) {
               Console.WriteLine("File: {0}", fn);
               Console.WriteLine("The file does not exist.\n");
            }
            else {
               try {
                  AssemblyName an = AssemblyName.GetAssemblyName(fn);
                  Console.WriteLine("Assembly: {0}", an.Name);
                  if (an.ProcessorArchitecture ==
ProcessorArchitecture.MSIL)
                     Console.WriteLine("Architecture: AnyCPU");
                  else
                     Console.WriteLine("Architecture: {0}",
an.ProcessorArchitecture);

                  Console.WriteLine();
               }
               catch (BadImageFormatException) {
                  Console.WriteLine("File: {0}", fn);
                  Console.WriteLine("Not a valid assembly.\n");
               }
            }
         }
      }
   }
```

- Reflecting on C++ executable files may throw this exception. This is most likely caused by the C++ compiler stripping the relocation addresses or the .Reloc section from the executable file. To preserve the .relocation address in a C++ executable file, specify /fixed:no when linking.

BadImageFormatException uses the HRESULT `COR_E_BADIMAGEFORMAT`, which has the value 0x8007000B.

For a list of initial property values for an instance of BadImageFormatException, see the BadImageFormatException constructors.

Constructors

Expand table

BadImageFormatException()	Initializes a new instance of the BadImageFormatException class.
BadImageFormat Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the BadImageFormatException class with serialized data.

BadImageFormatException(String)	Initializes a new instance of the BadImageFormatException class with a specified error message.
BadImageFormatException(String, Exception)	Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.
BadImageFormatException(String, String)	Initializes a new instance of the BadImageFormatException class with a specified error message and file name.
BadImageFormatException(String, String, Exception)	Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
FileName	Gets the name of the file that causes this exception.
FusionLog	Gets the log file that describes why an assembly load failed.
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message and the name of the file that caused this exception.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the file name, assembly cache log, and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns the fully qualified name of this exception and possibly the error message, the name of the inner exception, and the stack trace.

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BadImageFormatException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the BadImageFormatException class.

Overloads

Expand table

BadImageFormatException()	Initializes a new instance of the BadImageFormatException class.
BadImageFormatException(String)	Initializes a new instance of the BadImageFormatException class with a specified error message.
BadImageFormatException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the BadImageFormatException class with serialized data.
BadImageFormatException(String, Exception)	Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.
BadImageFormatException(String, String)	Initializes a new instance of the BadImageFormatException class with a specified error message and file name.
BadImageFormatException(String, String, Exception)	Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

BadImageFormatException()

Initializes a new instance of the BadImageFormatException class.

```
C#

public BadImageFormatException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Format of the executable or library is invalid." This message takes into account the current system culture.

The following table shows the initial property values for an instance of BadImageFormatException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BadImageFormatException(String)

Initializes a new instance of the BadImageFormatException class with a specified error message.

```
C#
public BadImageFormatException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of BadImageFormatException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BadImageFormatException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the BadImageFormatException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected BadImageFormatException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

BadImageFormatException(String, Exception)

Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public BadImageFormatException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of BadImageFormatException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BadImageFormatException(String, String)

Initializes a new instance of the BadImageFormatException class with a specified error message and file name.

C#

```
public BadImageFormatException (string? message, string? fileName);
```

Parameters

`message` String

A message that describes the error.

`fileName` String

The full name of the file with the invalid image.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Expand table

Property	Value
FileName	The full name of the file with the invalid image.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

BadImageFormatException(String, String, Exception)

Initializes a new instance of the BadImageFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public BadImageFormatException (string? message, string? fileName,
Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`fileName` String

The full name of the file with the invalid image.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of BadImageFormatException.

Expand table

Property	Value
InnerException	The inner exception reference.
FileName	The full name of the file with the invalid image.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BadImageFormatException.FileName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the file that causes this exception.

```
C#
public string? FileName { get; }
```

Property Value

String

The name of the file with the invalid image, or a null reference if no file name was passed to the constructor for the current instance.

Remarks

This property returns the name of the file supplied to the constructor for the current instance, if any. If the file name was not specified or is a null reference (`Nothing` in Visual Basic), this property returns a null reference.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BadImageFormatException.FusionLog Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the log file that describes why an assembly load failed.

C#

```
public string? FusionLog { get; }
```

Property Value

String

A `String` containing errors reported by the assembly cache.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

BadImageFormatException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the error message and the name of the file that caused this exception.

```
C#
public override string Message { get; }
```

Property Value

String

A string containing the error message and the name of the file that caused this exception.

Remarks

If no message was supplied to the constructor for the current exception, this property returns a system-supplied error message. If the FileName property is not a null reference (`Nothing` in Visual Basic), the message includes the file name.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BadImageFormatException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the file name, assembly cache log, and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

SecurityException

The caller does not have the required permission.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BadImageFormatException.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the fully qualified name of this exception and possibly the error message, the name of the inner exception, and the stack trace.

C#

```
public override string ToString ();
```

Returns

String

A string containing the fully qualified name of this exception and possibly the error message, the name of the inner exception, and the stack trace.

Remarks

The string representation returned by this method includes the name of the exception, the value of the Message property, the result of calling `ToString` on the inner exception, the value of the FileName property, and the result of calling StackTrace. If any of these members is a null reference (`Nothing` in Visual Basic), its value is not included in the returned string.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Base64FormattingOptions Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies whether relevant ToBase64CharArray and ToBase64String methods insert line breaks in their output.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum Base64FormattingOptions
```

Inheritance Object → ValueType → Enum → Base64FormattingOptions

Attributes FlagsAttribute

Fields

Expand table

InsertLineBreaks	1	Inserts line breaks after every 76 characters in the string representation.
None	0	Does not insert line breaks after every 76 characters in the string representation.

Examples

The following example calls the Convert.ToBase64String(Byte[], Base64FormattingOptions) method with a `InsertLineBreaks` argument to insert line breaks in the string that is produced by encoding a 100-element byte array:

C#

```
using System;

public class Example
{
   public static void Main()
   {
       // Define a byte array.
       var bytes = new byte[100];
       int originalTotal = 0;
       for (int ctr = 0; ctr <= bytes.GetUpperBound(0); ctr++) {
          bytes[ctr] = (byte)(ctr + 1);
          originalTotal += bytes[ctr];
       }
       // Display summary information about the array.
       Console.WriteLine("The original byte array:");
       Console.WriteLine("   Total elements: {0}", bytes.Length);
       Console.WriteLine("   Length of String Representation: {0}",
                         BitConverter.ToString(bytes).Length);
       Console.WriteLine("   Sum of elements: {0:N0}", originalTotal);
       Console.WriteLine();

       // Convert the array to a base 64 string.
       string s = Convert.ToBase64String(bytes,

Base64FormattingOptions.InsertLineBreaks);
       Console.WriteLine("The base 64 string:\n   {0}\n", s);

       // Restore the byte array.
       Byte[] newBytes = Convert.FromBase64String(s);
       int newTotal = 0;
       foreach (var newByte in newBytes)
          newTotal += newByte;

       // Display summary information about the restored array.
       Console.WriteLine("   Total elements: {0}", newBytes.Length);
       Console.WriteLine("   Length of String Representation: {0}",
                         BitConverter.ToString(newBytes).Length);
       Console.WriteLine("   Sum of elements: {0:N0}", newTotal);
   }
}
// The example displays the following output:
//   The original byte array:
//      Total elements: 100
//      Length of String Representation: 299
//      Sum of elements: 5,050
//
//   The base 64 string:
//
AQIDBAUGBwgJCgsMDQ4PEBESExQVFhcYGRobHB0eHyAhIiMkJSYnKCkqKywtLi8wMTIzNDU2Nzg5
//   Ojs8PT4/QEFCQ0RFRkdISUpLTE1OT1BRUlNUVVZXWFlaW1xdXl9gYWJjZA==
//
//      Total elements: 100
```

```
//      Length of String Representation: 299
//      Sum of elements: 5,050
```

As the output from the example shows, the Convert.FromBase64String succeeds in restoring the original byte array; the line break characters are ignored during the conversion.

Remarks

The Convert.ToBase64CharArray and Convert.ToBase64String methods convert the value of an array of 8-bit unsigned integers to an equivalent string representation that consists of base 64 digits. The string representation can contain one or more line breaks, where a line break is defined as a carriage return character (U+000D) followed by a line feed character (U+000A). Because line breaks are considered white-space characters in a base-64 encoding, they are ignored when converting a base-64 encoded string back to a byte array. The line breaks are simply convenient when displaying the encoded string to a control or a device such as a console window.

The `None` and `InsertLineBreaks` values are mutually exclusive. Therefore, although the `Base64FormattingOptions` enumeration is marked with the FlagsAttribute attribute, it makes no sense to perform a bitwise combination of these two values.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

BitConverter Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts base data types to an array of bytes, and an array of bytes to base data types.

C#

```
public static class BitConverter
```

Inheritance Object → BitConverter

Examples

The following code example illustrates the use of several BitConverter class methods.

C#

```
// Example of BitConverter class methods.
using System;

class BitConverterDemo
{
    public static void Main( )
    {
        const string formatter = "{0,25}{1,30}";

        double  aDoubl  = 0.1111111111111111111;
        float   aSingl  = 0.1111111111111111111F;
        long    aLong   = 1111111111111111111;
        int     anInt   = 1111111111;
        short   aShort  = 11111;
        char    aChar   = '*';
        bool    aBool   = true;

        Console.WriteLine(
            "This example of methods of the BitConverter class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( formatter, "argument", "byte array" );
        Console.WriteLine( formatter, "--------", "----------" );
```

```
        // Convert values to Byte arrays and display them.
        Console.WriteLine( formatter, aDoubl,
            BitConverter.ToString( BitConverter.GetBytes( aDoubl ) ) );
        Console.WriteLine( formatter, aSingl,
            BitConverter.ToString( BitConverter.GetBytes( aSingl ) ) );
        Console.WriteLine( formatter, aLong,
            BitConverter.ToString( BitConverter.GetBytes( aLong ) ) );
        Console.WriteLine( formatter, anInt,
            BitConverter.ToString( BitConverter.GetBytes( anInt ) ) );
        Console.WriteLine( formatter, aShort,
            BitConverter.ToString( BitConverter.GetBytes( aShort ) ) );
        Console.WriteLine( formatter, aChar,
            BitConverter.ToString( BitConverter.GetBytes( aChar ) ) );
        Console.WriteLine( formatter, aBool,
            BitConverter.ToString( BitConverter.GetBytes( aBool ) ) );
    }
}

/*
This example of methods of the BitConverter class
generates the following output.

                 argument                    byte array
                 --------                    ----------
        0.111111111111111       1C-C7-71-1C-C7-71-BC-3F
                0.1111111                   39-8E-E3-3D
      1111111111111111111       C7-71-C4-2B-AB-75-6B-0F
               1111111111                   C7-35-3A-42
                    11111                         67-2B
                        *                         2A-00
                     True                            01
*/
```

Remarks

The BitConverter class helps manipulate value types in their fundamental form, as a series of bytes. A byte is defined as an 8-bit unsigned integer. The BitConverter class includes static methods to convert each of the primitive types to and from an array of bytes, as the following table illustrates.

Expand table

Type	To byte conversion	From byte conversion
Boolean	GetBytes(Boolean)	ToBoolean
Char	GetBytes(Char)	ToChar
Double	GetBytes(Double)	ToDouble

Type	To byte conversion	From byte conversion
	-or- DoubleToInt64Bits(Double)	-or- Int64BitsToDouble
Int16	GetBytes(Int16)	ToInt16
Int32	GetBytes(Int32)	ToInt32
Int64	GetBytes(Int64)	ToInt64
Single	GetBytes(Single)	ToSingle
UInt16	GetBytes(UInt16)	ToUInt16
UInt32	GetBytes(UInt32)	ToUInt32
UInt64	GetBytes(UInt64)	ToUInt64

If you use BitConverter methods to round-trip data, make sure that the GetBytes overload and the `To`*Type* method specify the same type. As the following example illustrates, restoring an array that represents a signed integer by calling the ToUInt32 method can result in a value that is different from the original. For more information, see Working with Signed Non-Decimal and Bitwise Values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int value = -16;
      Byte[] bytes = BitConverter.GetBytes(value);

      // Convert bytes back to int.
      int intValue = BitConverter.ToInt32(bytes, 0);
      Console.WriteLine("{0} = {1}: {2}",
                        value, intValue,
                        value.Equals(intValue) ? "Round-trips" : "Does not
round-trip");
      // Convert bytes to UInt32.
      uint uintValue = BitConverter.ToUInt32(bytes, 0);
      Console.WriteLine("{0} = {1}: {2}", value, uintValue,
                        value.Equals(uintValue) ? "Round-trips" : "Does not
round-trip");
   }
}
// The example displays the following output:
```

```
//        -16 = -16: Round-trips
//        -16 = 4294967280: Does not round-trip
```

The order of bytes in the array returned by the GetBytes method overloads (as well as the order of bits in the integer returned by the DoubleToInt64Bits method) depends on whether the computer architecture is little-endian or big-endian. Similarly, the order of bytes in the array and returned by the `To`*IntegerValue* methods and the ToChar method depends on whether the computer architecture is little-endian or big-endian. The endianness of an architecture is indicated by the IsLittleEndian property, which returns `true` on little-endian systems and `false` on big-endian systems. On little-endian systems, lower-order bytes precede higher-order bytes. On big-endian system, higher-order bytes precede lower-order bytes. The following table illustrates the difference in the byte arrays that result from passing the integer 1,234,567,890 (0x499602D2) to the GetBytes(Int32) method. The bytes are listed in order from the byte at index 0 to the byte at index 3.

Expand table

Little-endian	D2-02-96-49
Big-endian	49-96-02-D2

Because the return value of some methods depends on system architecture, be careful when transmitting byte data beyond machine boundaries:

- If all systems sending and receiving data are guaranteed to have the same endianness, nothing has be done to the data.
- If systems sending and receiving data can have different endianness, always transmit data in a particular order. This means that the order of bytes in the array may have to be reversed either before sending them or after receiving them. A common convention is to transmit data in network byte order (big-endian order). The following example provides an implementation for sending an integer value in network byte order.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int value = 12345678;
```

```
        byte[] bytes = BitConverter.GetBytes(value);
        Console.WriteLine(BitConverter.ToString(bytes));

        if (BitConverter.IsLittleEndian)
           Array.Reverse(bytes);

        Console.WriteLine(BitConverter.ToString(bytes));
        // Call method to send byte stream across machine boundaries.

        // Receive byte stream from beyond machine boundaries.
        Console.WriteLine(BitConverter.ToString(bytes));
        if (BitConverter.IsLittleEndian)
           Array.Reverse(bytes);

        Console.WriteLine(BitConverter.ToString(bytes));
        int result = BitConverter.ToInt32(bytes, 0);
        Console.WriteLine("Original value: {0}", value);
        Console.WriteLine("Returned value: {0}", result);
    }
}
// The example displays the following output on a little-endian system:
//       4E-61-BC-00
//       00-BC-61-4E
//       00-BC-61-4E
//       4E-61-BC-00
//       Original value: 12345678
//       Returned value: 12345678
```

- If systems sending and receiving data can have different endianness and the data to be transmitted consists of signed integers, call the IPAddress.HostToNetworkOrder method to convert the data to network byte order and the IPAddress.NetworkToHostOrder method to convert it to the order required by the recipient.

Fields

Expand table

IsLittleEndian	Indicates the byte order ("endianness") in which data is stored in this computer architecture.

Methods

Expand table

DoubleToInt64Bits(Double)	Converts the specified double-precision floating point number to a 64-bit signed integer.
DoubleToUInt64Bits(Double)	Converts the specified double-precision floating point number to a 64-bit unsigned integer.
GetBytes(Boolean)	Returns the specified Boolean value as a byte array.
GetBytes(Char)	Returns the specified Unicode character value as an array of bytes.
GetBytes(Double)	Returns the specified double-precision floating-point value as an array of bytes.
GetBytes(Half)	Returns the specified half-precision floating-point value as an array of bytes.
GetBytes(Int16)	Returns the specified 16-bit signed integer value as an array of bytes.
GetBytes(Int32)	Returns the specified 32-bit signed integer value as an array of bytes.
GetBytes(Int64)	Returns the specified 64-bit signed integer value as an array of bytes.
GetBytes(Single)	Returns the specified single-precision floating point value as an array of bytes.
GetBytes(UInt16)	Returns the specified 16-bit unsigned integer value as an array of bytes.
GetBytes(UInt32)	Returns the specified 32-bit unsigned integer value as an array of bytes.
GetBytes(UInt64)	Returns the specified 64-bit unsigned integer value as an array of bytes.
HalfToInt16Bits(Half)	Converts a half-precision floating-point value into a 16-bit integer.
HalfToUInt16Bits(Half)	Converts the specified half-precision floating point number to a 16-bit unsigned integer.
Int16BitsToHalf(Int16)	Reinterprets the specified 16-bit signed integer value as a half-precision floating-point value.
Int32BitsToSingle(Int32)	Reinterprets the specified 32-bit integer as a single-precision floating-point value.
Int64BitsToDouble(Int64)	Reinterprets the specified 64-bit signed integer to a double-precision floating point number.
SingleToInt32Bits(Single)	Converts a single-precision floating-point value into an integer.

SingleToUInt32Bits(Single)	Converts the specified single-precision floating point number to a 32-bit unsigned integer.
ToBoolean(Byte[], Int32)	Returns a Boolean value converted from the byte at a specified position in a byte array.
ToBoolean(ReadOnlySpan<Byte>)	Converts a read-only byte span to a Boolean value.
ToChar(Byte[], Int32)	Returns a Unicode character converted from two bytes at a specified position in a byte array.
ToChar(ReadOnlySpan<Byte>)	Converts a read-only byte span into a character.
ToDouble(Byte[], Int32)	Returns a double-precision floating point number converted from eight bytes at a specified position in a byte array.
ToDouble(ReadOnlySpan<Byte>)	Converts a read-only byte span into a double-precision floating-point value.
ToHalf(Byte[], Int32)	Returns a laft-precision floating point number converted from two bytes at a specified position in a byte array.
ToHalf(ReadOnlySpan<Byte>)	Converts a read-only byte span into a half-precision floating-point value.
ToInt16(Byte[], Int32)	Returns a 16-bit signed integer converted from two bytes at a specified position in a byte array.
ToInt16(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 16-bit signed integer.
ToInt32(Byte[], Int32)	Returns a 32-bit signed integer converted from four bytes at a specified position in a byte array.
ToInt32(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 32-bit signed integer.
ToInt64(Byte[], Int32)	Returns a 64-bit signed integer converted from eight bytes at a specified position in a byte array.
ToInt64(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 64-bit signed integer.
ToSingle(Byte[], Int32)	Returns a single-precision floating point number converted from four bytes at a specified position in a byte array.
ToSingle(ReadOnlySpan<Byte>)	Converts a read-only byte span into a single-precision floating-point value.

ToString(Byte[])	Converts the numeric value of each element of a specified array of bytes to its equivalent hexadecimal string representation.
ToString(Byte[], Int32)	Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.
ToString(Byte[], Int32, Int32)	Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.
ToUInt16(Byte[], Int32)	Returns a 16-bit unsigned integer converted from two bytes at a specified position in a byte array.
ToUInt16(ReadOnlySpan<Byte>)	Converts a read-only byte-span into a 16-bit unsigned integer.
ToUInt32(Byte[], Int32)	Returns a 32-bit unsigned integer converted from four bytes at a specified position in a byte array.
ToUInt32(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 32-bit unsigned integer.
ToUInt64(Byte[], Int32)	Returns a 64-bit unsigned integer converted from eight bytes at a specified position in a byte array.
ToUInt64(ReadOnlySpan<Byte>)	Converts bytes into an unsigned long.
TryWriteBytes(Span<Byte>, Boolean)	Converts a Boolean into a span of bytes.
TryWriteBytes(Span<Byte>, Char)	Converts a character into a span of bytes.
TryWriteBytes(Span<Byte>, Double)	Converts a double-precision floating-point value into a span of bytes.
TryWriteBytes(Span<Byte>, Half)	Converts a half-precision floating-point value into a span of bytes.
TryWriteBytes(Span<Byte>, Int16)	Converts a 16-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Int32)	Converts a 32-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Int64)	Converts a 64-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Single)	Converts a single-precision floating-point value into a span of bytes.

TryWriteBytes(Span<Byte>, UInt16)	Converts an unsigned 16-bit integer into a span of bytes.
TryWriteBytes(Span<Byte>, UInt32)	Converts a 32-bit unsigned integer into a span of bytes.
TryWriteBytes(Span<Byte>, UInt64)	Converts an unsigned 64-bit integer into a span of bytes.
UInt16BitsToHalf(UInt16)	Converts the specified 16-bit unsigned integer to a half-precision floating point number.
UInt32BitsToSingle(UInt32)	Converts the specified 32-bit unsigned integer to a single-precision floating point number.
UInt64BitsToDouble(UInt64)	Converts the specified 64-bit unsigned integer to a double-precision floating point number.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Byte

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.IsLittleEndian Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates the byte order ("endianness") in which data is stored in this computer architecture.

C#

```
public static readonly bool IsLittleEndian;
```

Field Value

Boolean

Examples

The following code example illustrates the use of the `IsLittleEndian` field.

C#

```
// Example of the BitConverter.IsLittleEndian field.
using System;

class LittleEndDemo
{
    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.IsLittleEndian field " +
            "generates \nthe following output when run on " +
            "x86-class computers.\n");
        Console.WriteLine( "IsLittleEndian:  {0}",
            BitConverter.IsLittleEndian );
    }
}

/*
This example of the BitConverter.IsLittleEndian field generates
the following output when run on x86-class computers.
```

```
IsLittleEndian:  True
*/
```

Remarks

Different computer architectures store data using different byte orders. "Big-endian" means the most significant byte is on the left end of a word. "Little-endian" means the most significant byte is on the right end of a word.

> **Note**
>
> You can convert from network byte order to the byte order of the host computer without retrieving the value of the **BitConverter.IsLittleEndian** field by passing a 16-bit, 32-bit, or 64 bit integer to the **IPAddress.HostToNetworkOrder** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.DoubleToInt64Bits(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified double-precision floating point number to a 64-bit signed integer.

```
C#

public static long DoubleToInt64Bits (double value);
```

Parameters

`value` Double

The number to convert.

Returns

Int64

A 64-bit signed integer whose value is equivalent to `value`.

Examples

The following code example converts the bit patterns of several Double values to Int64 values with the `DoubleToInt64Bits` method.

```
C#

// Example of the BitConverter.DoubleToInt64Bits method.
using System;

class DoubleToInt64BitsDemo
{
    const string formatter = "{0,25:E16}{1,23:X16}";
```

```
    // Reinterpret the double argument as a long.
    public static void DoubleToLongBits( double argument )
    {
        long longValue;
        longValue = BitConverter.DoubleToInt64Bits( argument );

        // Display the resulting long in hexadecimal.
        Console.WriteLine( formatter, argument, longValue );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.DoubleToInt64Bits( " +
            "double ) \nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "double argument",
            "hexadecimal value" );
        Console.WriteLine( formatter, "---------------",
            "-----------------" );

        // Convert double values and display the results.
        DoubleToLongBits( 1.0 );
        DoubleToLongBits( 15.0 );
        DoubleToLongBits( 255.0 );
        DoubleToLongBits( 4294967295.0 );
        DoubleToLongBits( 0.00390625 );
        DoubleToLongBits( 0.0000000002328306436538696289062 5 );
        DoubleToLongBits( 1.234567890123E-300 );
        DoubleToLongBits( 1.23456789012345E-150 );
        DoubleToLongBits( 1.2345678901234565 );
        DoubleToLongBits( 1.2345678901234567 );
        DoubleToLongBits( 1.2345678901234569 );
        DoubleToLongBits( 1.23456789012345678E+150 );
        DoubleToLongBits( 1.234567890123456789E+300 );
        DoubleToLongBits( double.MinValue );
        DoubleToLongBits( double.MaxValue );
        DoubleToLongBits( double.Epsilon );
        DoubleToLongBits( double.NaN );
        DoubleToLongBits( double.NegativeInfinity );
        DoubleToLongBits( double.PositiveInfinity );
    }
}

/*
This example of the BitConverter.DoubleToInt64Bits( double )
method generates the following output.

          double argument      hexadecimal value
          ---------------      -----------------
  1.0000000000000000E+000       3FF0000000000000
  1.5000000000000000E+001       402E000000000000
  2.5500000000000000E+002       406FE00000000000
  4.2949672950000000E+009       41EFFFFFFFE00000
  3.9062500000000000E-003       3F70000000000000
  2.3283064365386963E-010       3DF0000000000000
```

```
  1.2345678901230000E-300       01AA74FE1C1E7E45
  1.2345678901234500E-150       20D02A36586DB4BB
  1.2345678901234565E+000       3FF3C0CA428C59FA
  1.2345678901234567E+000       3FF3C0CA428C59FB
  1.2345678901234569E+000       3FF3C0CA428C59FC
  1.2345678901234569E+150       5F182344CD3CDF9F
  1.2345678901234569E+300       7E3D7EE8BCBBD352
 -1.7976931348623157E+308       FFEFFFFFFFFFFFFF
  1.7976931348623157E+308       7FEFFFFFFFFFFFFF
  4.9406564584124654E-324       0000000000000001
                     NaN        FFF8000000000000
               -Infinity        FFF0000000000000
                Infinity        7FF0000000000000
*/
```

Remarks

The order of bits in the integer returned by the DoubleToInt64Bits method depends on whether the computer architecture is little-endian or big-endian.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64BitsToDouble(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

BitConverter.DoubleToUInt64Bits(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified double-precision floating point number to a 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong DoubleToUInt64Bits (double value);
```

Parameters

`value` Double

The number to convert.

Returns

UInt64

A 64-bit unsigned integer whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.GetBytes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified data to an array of bytes.

Overloads

Expand table

GetBytes(Boolean)	Returns the specified Boolean value as a byte array.
GetBytes(Char)	Returns the specified Unicode character value as an array of bytes.
GetBytes(Double)	Returns the specified double-precision floating-point value as an array of bytes.
GetBytes(Half)	Returns the specified half-precision floating-point value as an array of bytes.
GetBytes(Int16)	Returns the specified 16-bit signed integer value as an array of bytes.
GetBytes(Int32)	Returns the specified 32-bit signed integer value as an array of bytes.
GetBytes(Int64)	Returns the specified 64-bit signed integer value as an array of bytes.
GetBytes(Single)	Returns the specified single-precision floating point value as an array of bytes.
GetBytes(UInt16)	Returns the specified 16-bit unsigned integer value as an array of bytes.
GetBytes(UInt32)	Returns the specified 32-bit unsigned integer value as an array of bytes.
GetBytes(UInt64)	Returns the specified 64-bit unsigned integer value as an array of bytes.

GetBytes(Boolean)

Returns the specified Boolean value as a byte array.

```
C#
```

```
public static byte[] GetBytes (bool value);
```

Parameters

`value` Boolean

The value to convert.

Returns

Byte[]

A byte array with length 1.

Examples

The following example converts the bit patterns of Boolean values to Byte arrays with the `GetBytes` method.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      // Define Boolean true and false values.
      bool[] values = { true, false };

      // Display the value and its corresponding byte array.
      Console.WriteLine("{0,10}{1,16}\n", "Boolean", "Bytes");
      foreach (var value in values) {
         byte[] bytes = BitConverter.GetBytes(value);
         Console.WriteLine("{0,10}{1,16}", value,
                           BitConverter.ToString(bytes));
      }
   }
}
// The example displays the following output:
//        Boolean           Bytes
//
//           True              01
//          False              00
```

Remarks

You can convert a byte array back to a Boolean value by calling the ToBoolean method.

See also

- ToBoolean(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Char)

Returns the specified Unicode character value as an array of bytes.

C#

```
public static byte[] GetBytes (char value);
```

Parameters

`value` Char

The character to convert.

Returns

Byte[]

An array of bytes with length 2.

Examples

The following code example converts the bit patterns of Char values (Unicode characters) to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( char ) method.
using System;

class GetBytesCharDemo
{
    const string formatter = "{0,10}{1,16}";

    // Convert a char argument to a byte array and display it.
    public static void GetBytesChar( char argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( char ) " +
            "\nmethod generates the following output.\r\n" );
        Console.WriteLine( formatter, "char", "byte array" );
        Console.WriteLine( formatter, "----", "----------" );

        // Convert char values and display the results.
        GetBytesChar( '\0' );
        GetBytesChar( ' ' );
        GetBytesChar( '*' );
        GetBytesChar( '3' );
        GetBytesChar( 'A' );
        GetBytesChar( '[' );
        GetBytesChar( 'a' );
        GetBytesChar( '{' );
    }
}

/*
This example of the BitConverter.GetBytes( char )
method generates the following output.

      char      byte array
      ----      ----------
                     00-00
                     20-00
         *           2A-00
         3           33-00
         A           41-00
```

```
        [           5B-00
        a           61-00
        {           7B-00
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToChar(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Double)

Returns the specified double-precision floating-point value as an array of bytes.

C#

```
public static byte[] GetBytes (double value);
```

Parameters

`value` Double

The number to convert.

Returns

Byte[]

An array of bytes with length 8.

Examples

The following code example converts the bit patterns of Double values to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( double ) method.
using System;

class GetBytesDoubleDemo
{
    const string formatter = "{0,25:E16}{1,30}";

    // Convert a double argument to a byte array and display it.
    public static void GetBytesDouble( double argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( double ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "double", "byte array" );
        Console.WriteLine( formatter, "------", "----------" );

        // Convert double values and display the results.
        GetBytesDouble( 0.0 );
        GetBytesDouble( 1.0 );
        GetBytesDouble( 255.0 );
        GetBytesDouble( 4294967295.0 );
        GetBytesDouble( 0.00390625 );
        GetBytesDouble( 0.0000000002328306436538696289062 5 );
        GetBytesDouble( 1.23456789012345E-300 );
        GetBytesDouble( 1.2345678901234565 );
        GetBytesDouble( 1.2345678901234567 );
        GetBytesDouble( 1.2345678901234569 );
        GetBytesDouble( 1.23456789012345678E+300 );
        GetBytesDouble( double.MinValue );
        GetBytesDouble( double.MaxValue );
        GetBytesDouble( double.Epsilon );
        GetBytesDouble( double.NaN );
```

```
        GetBytesDouble( double.NegativeInfinity );
        GetBytesDouble( double.PositiveInfinity );
    }
}

/*
This example of the BitConverter.GetBytes( double )
method generates the following output.

                   double                    byte array
                   ------                    ----------
  0.0000000000000000E+000       00-00-00-00-00-00-00-00
  1.0000000000000000E+000       00-00-00-00-00-00-F0-3F
  2.5500000000000000E+002       00-00-00-00-00-E0-6F-40
  4.2949672950000000E+009       00-00-E0-FF-FF-FF-EF-41
  3.9062500000000000E-003       00-00-00-00-00-00-70-3F
  2.3283064365386963E-010       00-00-00-00-00-00-F0-3D
  1.2345678901234500E-300       DF-88-1E-1C-FE-74-AA-01
  1.2345678901234565E+000       FA-59-8C-42-CA-C0-F3-3F
  1.2345678901234567E+000       FB-59-8C-42-CA-C0-F3-3F
  1.2345678901234569E+000       FC-59-8C-42-CA-C0-F3-3F
  1.2345678901234569E+300       52-D3-BB-BC-E8-7E-3D-7E
 -1.7976931348623157E+308       FF-FF-FF-FF-FF-FF-EF-FF
  1.7976931348623157E+308       FF-FF-FF-FF-FF-FF-EF-7F
  4.9406564584124654E-324       01-00-00-00-00-00-00-00
                     NaN       00-00-00-00-00-00-F8-FF
               -Infinity       00-00-00-00-00-00-F0-FF
                Infinity       00-00-00-00-00-00-F0-7F
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToDouble(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Half)

Returns the specified half-precision floating-point value as an array of bytes.

C#

```
public static byte[] GetBytes (Half value);
```

Parameters

`value` Half

The number to convert.

Returns

Byte[]

An array of bytes with length 2.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

GetBytes(Int16)

Returns the specified 16-bit signed integer value as an array of bytes.

C#

```
public static byte[] GetBytes (short value);
```

Parameters

`value` Int16

The number to convert.

Returns

Byte[]

An array of bytes with length 2.

Examples

The following code example converts the bit patterns of Int16 values to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( short ) method.
using System;

class GetBytesInt16Demo
{
    const string formatter = "{0,10}{1,13}";

    // Convert a short argument to a byte array and display it.
    public static void GetBytesInt16( short argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( short ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "short", "byte array" );
        Console.WriteLine( formatter, "-----", "----------" );

        // Convert short values and display the results.
        GetBytesInt16( 0 );
        GetBytesInt16( 15 );
        GetBytesInt16( -15 );
        GetBytesInt16( 10000 );
```

```
        GetBytesInt16( -10000 );
        GetBytesInt16( short.MinValue );
        GetBytesInt16( short.MaxValue );
    }
}

/*
This example of the BitConverter.GetBytes( short )
method generates the following output.

     short   byte array
     -----   ----------
         0        00-00
        15        0F-00
       -15        F1-FF
     10000        10-27
    -10000        F0-D8
    -32768        00-80
     32767        FF-7F
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToInt16(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Int32)

Returns the specified 32-bit signed integer value as an array of bytes.

C#

```
public static byte[] GetBytes (int value);
```

Parameters

`value` Int32

The number to convert.

Returns

Byte[]

An array of bytes with length 4.

Examples

The following code example converts the bit patterns of Int32 values to Byte arrays with the `GetBytes` method.

C#

```
using System;

class Example
{
    public static void Main( )
    {
        // Define an array of integers.
        int[] values = { 0, 15, -15, 0x100000,  -0x100000, 1000000000,
                         -1000000000, int.MinValue, int.MaxValue };

        // Convert each integer to a byte array.
        Console.WriteLine("{0,16}{1,10}{2,17}", "Integer",
                          "Endian", "Byte Array");
        Console.WriteLine("{0,16}{1,10}{2,17}", "---", "------",
                          "----------" );
        foreach (var value in values) {
          byte[] byteArray = BitConverter.GetBytes(value);
          Console.WriteLine("{0,16}{1,10}{2,17}", value,
                            BitConverter.IsLittleEndian ? "Little" : "
Big",
                            BitConverter.ToString(byteArray));
```

```
        }
    }
}
// This example displays output like the following:
//              Integer    Endian       Byte Array
//                  ---    ------       ----------
//                    0    Little      00-00-00-00
//                   15    Little      0F-00-00-00
//                  -15    Little      F1-FF-FF-FF
//              1048576    Little      00-00-10-00
//             -1048576    Little      00-00-F0-FF
//           1000000000    Little      00-CA-9A-3B
//          -1000000000    Little      00-36-65-C4
//          -2147483648    Little      00-00-00-80
//           2147483647    Little      FF-FF-FF-7F
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToInt32(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Int64)

Returns the specified 64-bit signed integer value as an array of bytes.

C#

```
public static byte[] GetBytes (long value);
```

Parameters

`value` Int64

The number to convert.

Returns

Byte[]

An array of bytes with length 8.

Examples

The following example calls the GetBytes method to convert each element in an Int64 array to a Byte arrays.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of Int64 values.
        long[] values = { 0, 0xFFFFFF, -0xFFFFFF, 1000000000,
-1000000000,
                          0x100000000, -0x100000000, 0xAAAAAAAAAAAA,
                          -0xAAAAAAAAAAAA, 1000000000000000000,
                          -1000000000000000000, long.MinValue,
                          long.MaxValue };

        Console.WriteLine( "{0,22}{1,10} {2,30}", "Int64", "Endian",
"Byte Array");
        Console.WriteLine( "{0,22}{1,10} {2,30}", "----", "------", "----
------" );

        foreach (var value in values) {
            // Convert each Int64 value to a byte array.
            byte[] byteArray = BitConverter.GetBytes(value);
            // Display the result.
            Console.WriteLine("{0,22}{1,10}{2,30}", value,
                              BitConverter.IsLittleEndian ? "Little" :
"Big",
```

```
                                BitConverter.ToString(byteArray));
        }
    }
}
// The example displays output like the following:
//                     Int64    Endian                     Byte Array
//                      ----    ------                     ----------
//                         0    Little       00-00-00-00-00-00-00-00
//                  16777215    Little       FF-FF-FF-00-00-00-00-00
//                 -16777215    Little       01-00-00-FF-FF-FF-FF-FF
//                1000000000    Little       00-CA-9A-3B-00-00-00-00
//               -1000000000    Little       00-36-65-C4-FF-FF-FF-FF
//                4294967296    Little       00-00-00-00-01-00-00-00
//               -4294967296    Little       00-00-00-00-FF-FF-FF-FF
//           187649984473770    Little       AA-AA-AA-AA-AA-AA-00-00
//          -187649984473770    Little       56-55-55-55-55-55-FF-FF
//       1000000000000000000    Little       00-00-64-A7-B3-B6-E0-0D
//      -1000000000000000000    Little       00-00-9C-58-4C-49-1F-F2
//      -9223372036854775808    Little       00-00-00-00-00-00-00-80
//       9223372036854775807    Little       FF-FF-FF-FF-FF-FF-FF-7F
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToInt64(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(Single)

Returns the specified single-precision floating point value as an array of bytes.

```
C#

public static byte[] GetBytes (float value);
```

Parameters

`value` Single

The number to convert.

Returns

Byte[]

An array of bytes with length 4.

Examples

The following code example converts the bit patterns of Single values to Byte arrays with the `GetBytes` method.

```
C#

// Example of the BitConverter.GetBytes( float ) method.
using System;

class GetBytesSingleDemo
{
    const string formatter = "{0,16:E7}{1,20}";

    // Convert a float argument to a byte array and display it.
    public static void GetBytesSingle( float argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( float ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "float", "byte array" );
```

```
        Console.WriteLine( formatter, "-----", "----------" );

        // Convert float values and display the results.
        GetBytesSingle( 0.0F );
        GetBytesSingle( 1.0F );
        GetBytesSingle( 15.0F );
        GetBytesSingle( 65535.0F );
        GetBytesSingle( 0.00390625F );
        GetBytesSingle( 0.00000000023283064365386962890625F );
        GetBytesSingle( 1.2345E-35F );
        GetBytesSingle( 1.2345671F );
        GetBytesSingle( 1.2345673F );
        GetBytesSingle( 1.2345677F );
        GetBytesSingle( 1.23456789E+35F );
        GetBytesSingle( float.MinValue );
        GetBytesSingle( float.MaxValue );
        GetBytesSingle( float.Epsilon );
        GetBytesSingle( float.NaN );
        GetBytesSingle( float.NegativeInfinity );
        GetBytesSingle( float.PositiveInfinity );
    }
}

/*
This example of the BitConverter.GetBytes( float )
method generates the following output.

           float          byte array
           -----          ----------
  0.0000000E+000         00-00-00-00
  1.0000000E+000         00-00-80-3F
  1.5000000E+001         00-00-70-41
  6.5535000E+004         00-FF-7F-47
  3.9062500E-003         00-00-80-3B
  2.3283064E-010         00-00-80-2F
  1.2345000E-035         49-46-83-05
  1.2345671E+000         4B-06-9E-3F
  1.2345673E+000         4D-06-9E-3F
  1.2345676E+000         50-06-9E-3F
  1.2345679E+035         1E-37-BE-79
 -3.4028235E+038         FF-FF-7F-FF
  3.4028235E+038         FF-FF-7F-7F
  1.4012985E-045         01-00-00-00
             NaN         00-00-C0-FF
       -Infinity         00-00-80-FF
        Infinity         00-00-80-7F
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToSingle(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(UInt16)

> Important
>
> This API is not CLS-compliant.

Returns the specified 16-bit unsigned integer value as an array of bytes.

C#

```
[System.CLSCompliant(false)]
public static byte[] GetBytes (ushort value);
```

Parameters

`value` UInt16

The number to convert.

Returns

Byte[]

An array of bytes with length 2.

Attributes CLSCompliantAttribute

Examples

The following code example converts the bit patterns of UInt16 values to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( ushort ) method.
using System;

class GetBytesUInt16Demo
{
    const string formatter = "{0,10}{1,13}";

    // Convert a ushort argument to a byte array and display it.
    public static void GetBytesUInt16( ushort argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( ushort ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "ushort", "byte array" );
        Console.WriteLine( formatter, "------", "----------" );

        // Convert ushort values and display the results.
        GetBytesUInt16( 15 );
        GetBytesUInt16( 1023 );
        GetBytesUInt16( 10000 );
        GetBytesUInt16( ushort.MinValue );
        GetBytesUInt16( (ushort)short.MaxValue );
        GetBytesUInt16( ushort.MaxValue );
    }
}

/*
This example of the BitConverter.GetBytes( ushort )
method generates the following output.

    ushort   byte array
    ------   ----------
        15       0F-00
      1023       FF-03
     10000       10-27
         0       00-00
```

```
     32767        FF-7F
     65535        FF-FF
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToUInt16(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(UInt32)

> Important
>
> This API is not CLS-compliant.

Returns the specified 32-bit unsigned integer value as an array of bytes.

C#

```
[System.CLSCompliant(false)]
public static byte[] GetBytes (uint value);
```

Parameters

`value` UInt32

The number to convert.

Returns

Byte[]

An array of bytes with length 4.

Attributes CLSCompliantAttribute

Examples

The following code example converts the bit patterns of UInt32 values to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( uint ) method.
using System;

class GetBytesUInt32Demo
{
    const string formatter = "{0,16}{1,20}";

    // Convert a uint argument to a byte array and display it.
    public static void GetBytesUInt32( uint argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( uint ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "uint", "byte array" );
        Console.WriteLine( formatter, "----", "----------" );

        // Convert uint values and display the results.
        GetBytesUInt32( 15 );
        GetBytesUInt32( 1023 );
        GetBytesUInt32( 0x100000 );
        GetBytesUInt32( 1000000000 );
        GetBytesUInt32( uint.MinValue );
        GetBytesUInt32( int.MaxValue );
```

```
        GetBytesUInt32( uint.MaxValue );
    }
}

/*
This example of the BitConverter.GetBytes( uint )
method generates the following output.

            uint          byte array
            ----          ----------
              15         0F-00-00-00
            1023         FF-03-00-00
         1048576         00-00-10-00
      1000000000         00-CA-9A-3B
               0         00-00-00-00
      2147483647         FF-FF-FF-7F
      4294967295         FF-FF-FF-FF
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToUInt32(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBytes(UInt64)

> Important
>
> This API is not CLS-compliant.

Returns the specified 64-bit unsigned integer value as an array of bytes.

C#

```
[System.CLSCompliant(false)]
public static byte[] GetBytes (ulong value);
```

Parameters

`value` UInt64

The number to convert.

Returns

Byte[]

An array of bytes with length 8.

Attributes CLSCompliantAttribute

Examples

The following code example converts the bit patterns of UInt64 values to Byte arrays with the `GetBytes` method.

C#

```
// Example of the BitConverter.GetBytes( ulong ) method.
using System;

class GetBytesUInt64Demo
{
    const string formatter = "{0,22}{1,30}";

    // Convert a ulong argument to a byte array and display it.
    public static void GetBytesUInt64( ulong argument )
    {
        byte[ ] byteArray = BitConverter.GetBytes( argument );
        Console.WriteLine( formatter, argument,
            BitConverter.ToString( byteArray ) );
```

```
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.GetBytes( ulong ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "ulong", "byte array" );
        Console.WriteLine( formatter, "-----", "----------" );

        // Convert ulong values and display the results.
        GetBytesUInt64( 0xFFFFFF );
        GetBytesUInt64( 1000000000 );
        GetBytesUInt64( 0x100000000 );
        GetBytesUInt64( 0xAAAAAAAAAAAA );
        GetBytesUInt64( 1000000000000000000 );
        GetBytesUInt64( 10000000000000000000 );
        GetBytesUInt64( ulong.MinValue );
        GetBytesUInt64( long.MaxValue );
        GetBytesUInt64( ulong.MaxValue );
    }
}

/*
This example of the BitConverter.GetBytes( ulong )
method generates the following output.

                 ulong                    byte array
                 -----                    ----------
              16777215       FF-FF-FF-00-00-00-00-00
            1000000000       00-CA-9A-3B-00-00-00-00
            4294967296       00-00-00-00-01-00-00-00
       187649984473770       AA-AA-AA-AA-AA-AA-00-00
   1000000000000000000       00-00-64-A7-B3-B6-E0-0D
  10000000000000000000       00-00-E8-89-04-23-C7-8A
                     0       00-00-00-00-00-00-00-00
   9223372036854775807       FF-FF-FF-FF-FF-FF-FF-7F
  18446744073709551615       FF-FF-FF-FF-FF-FF-FF-FF
*/
```

Remarks

The order of bytes in the array returned by the GetBytes method depends on whether the computer architecture is little-endian or big-endian.

See also

- ToUInt64(Byte[], Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.HalfToInt16Bits(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a half-precision floating-point value into a 16-bit integer.

C#

```
public static short HalfToInt16Bits (Half value);
```

Parameters

`value` Half

The half-precision floating-point value to convert.

Returns

Int16

An integer representing the converted half-precision floating-point value.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

BitConverter.HalfToUInt16Bits(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified half-precision floating point number to a 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort HalfToUInt16Bits (Half value);
```

Parameters

`value` Half

The number to convert.

Returns

UInt16

A 16-bit unsigned integer whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.Int16BitsToHalf(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reinterprets the specified 16-bit signed integer value as a half-precision floating-point value.

C#

```
public static Half Int16BitsToHalf (short value);
```

Parameters

`value` Int16

The 16-bit signed integer value to convert.

Returns

Half

A half-precision floating-point value that represents the converted integer.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

BitConverter.Int32BitsToSingle(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reinterprets the specified 32-bit integer as a single-precision floating-point value.

C#

```
public static float Int32BitsToSingle (int value);
```

Parameters

`value` Int32

The integer to convert.

Returns

Single

A single-precision floating-point value that represents the converted integer.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

BitConverter.Int64BitsToDouble(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reinterprets the specified 64-bit signed integer to a double-precision floating point number.

C#

```
public static double Int64BitsToDouble (long value);
```

Parameters

`value` Int64

The number to convert.

Returns

Double

A double-precision floating point number whose value is equivalent to `value`.

Examples

The following code example converts the bit patterns of several Int64 values to Double values with the `Int64BitsToDouble` method.

C#

```
// Example of the BitConverter.Int64BitsToDouble method.
using System;

class Int64BitsToDoubleDemo
{
    const string formatter = "{0,20}{1,27:E16}";
```

```
    // Reinterpret the long argument as a double.
    public static void LongBitsToDouble( long argument )
    {
        double doubleValue;
        doubleValue = BitConverter.Int64BitsToDouble( argument );

        // Display the argument in hexadecimal.
        Console.WriteLine( formatter,
            string.Format( "0x{0:X16}", argument ), doubleValue );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the BitConverter.Int64BitsToDouble( " +
            "long ) \nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "long argument",
            "double value" );
        Console.WriteLine( formatter, "-------------",
            "------------" );

        // Convert long values and display the results.
        LongBitsToDouble( 0 );
        LongBitsToDouble( 0x3FF0000000000000 );
        LongBitsToDouble( 0x402E000000000000 );
        LongBitsToDouble( 0x406FE00000000000 );
        LongBitsToDouble( 0x41EFFFFFFE00000 );
        LongBitsToDouble( 0x3F70000000000000 );
        LongBitsToDouble( 0x3DF0000000000000 );
        LongBitsToDouble( 0x0000000000000001 );
        LongBitsToDouble( 0x000000000000FFFF );
        LongBitsToDouble( 0x0000FFFFFFFFFFFF );
        LongBitsToDouble( unchecked( (long)0xFFFFFFFFFFFFFFFF ) );
        LongBitsToDouble( unchecked( (long)0xFFF0000000000000 ) );
        LongBitsToDouble( 0x7FF0000000000000 );
        LongBitsToDouble( unchecked( (long)0xFFEFFFFFFFFFFFFF ) );
        LongBitsToDouble( 0x7FEFFFFFFFFFFFFF );
        LongBitsToDouble( long.MinValue );
        LongBitsToDouble( long.MaxValue );
    }
}

/*
This example of the BitConverter.Int64BitsToDouble( long )
method generates the following output.

       long argument               double value
       -------------               ------------
  0x0000000000000000    0.0000000000000000E+000
  0x3FF0000000000000    1.0000000000000000E+000
  0x402E000000000000    1.5000000000000000E+001
  0x406FE00000000000    2.5500000000000000E+002
  0x41EFFFFFFE00000     4.2949672950000000E+009
  0x3F70000000000000    3.9062500000000000E-003
  0x3DF0000000000000    2.3283064365386963E-010
```

```
  0x0000000000000001    4.9406564584124654E-324
  0x000000000000FFFF    3.2378592100206092E-319
  0x0000FFFFFFFFFFFF    1.3906711615669959E-309
  0xFFFFFFFFFFFFFFFF                        NaN
  0xFFF0000000000000                  -Infinity
  0x7FF0000000000000                   Infinity
  0xFFEFFFFFFFFFFFFF   -1.7976931348623157E+308
  0x7FEFFFFFFFFFFFFF    1.7976931348623157E+308
  0x8000000000000000    0.0000000000000000E+000
  0x7FFFFFFFFFFFFFFF                        NaN
*/
```

Remarks

Typically, `value` is an integer that is returned by the DoubleToInt64Bits method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- DoubleToInt64Bits(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.SingleToInt32Bits(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a single-precision floating-point value into an integer.

C#

```
public static int SingleToInt32Bits (float value);
```

Parameters

`value` Single

The single-precision floating-point value to convert.

Returns

Int32

An integer representing the converted single-precision floating-point value.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

BitConverter.SingleToUInt32Bits(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified single-precision floating point number to a 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint SingleToUInt32Bits (float value);
```

Parameters

`value` Single

The number to convert.

Returns

UInt32

A 32-bit unsigned integer whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.ToBoolean Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToBoolean(ReadOnlySpan<Byte>)	Converts a read-only byte span to a Boolean value.
ToBoolean(Byte[], Int32)	Returns a Boolean value converted from the byte at a specified position in a byte array.

ToBoolean(ReadOnlySpan<Byte>)

Converts a read-only byte span to a Boolean value.

C#

```
public static bool ToBoolean (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Boolean

A Boolean representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToBoolean(Byte[], Int32)

Returns a Boolean value converted from the byte at a specified position in a byte array.

```
C#

public static bool ToBoolean (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

A byte array.

`startIndex` Int32

The index of the byte within `value` to convert.

Returns

Boolean

`true` if the byte at `startIndex` in `value` is nonzero; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Boolean values with the `ToBoolean` method.

C#

```
using System;

class Example
{
    public static void Main( )
    {
        // Define an array of byte values.
        byte[] bytes = { 0, 1, 2, 4, 8, 16, 32, 64, 128, 255 };

        Console.WriteLine("{0,5}{1,16}{2,10}\n", "index", "array
element", "bool" );
        // Convert each array element to a Boolean value.
        for (int index = 0; index < bytes.Length; index++)
           Console.WriteLine("{0,5}{1,16:X2}{2,10}", index, bytes[index],
                             BitConverter.ToBoolean(bytes, index));
    }
}
// The example displays the following output:
//     index   array element      bool
//
//         0              00     False
//         1              01      True
//         2              02      True
//         3              04      True
//         4              08      True
//         5              10      True
//         6              20      True
//         7              40      True
//         8              80      True
//         9              FF      True
```

Remarks

To convert a Boolean value to its byte representation, call the GetBytes(Boolean) method.

See also

- GetBytes(Boolean)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToChar Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToChar(Byte[], Int32)	Returns a Unicode character converted from two bytes at a specified position in a byte array.
ToChar(ReadOnlySpan<Byte>)	Converts a read-only byte span into a character.

ToChar(Byte[], Int32)

Returns a Unicode character converted from two bytes at a specified position in a byte array.

```
C#

public static char ToChar (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array that includes the two bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

Char

The character formed by two bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` equals the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Char values (Unicode characters) with the `ToChar` method.

C#

```
// Example of the BitConverter.ToChar method.
using System;

class BytesToCharDemo
{
    const string formatter = "{0,5}{1,17}{2,8}";

    // Convert two byte array elements to a char and display it.
    public static void BAToChar( byte[] bytes, int index )
    {
        char value = BitConverter.ToChar( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 2 ), value );
    }

    public static void Main( )
    {
        byte[] byteArray = {
             32,   0,   0,  42,   0,  65,   0, 125,   0,
            197,   0, 168,   3,  41,   4, 172,  32 };

        Console.WriteLine(
            "This example of the BitConverter.ToChar( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts \nelements of a byte array to char values.\n" );
        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );
        Console.WriteLine( BitConverter.ToString( byteArray ) );
        Console.WriteLine( );
        Console.WriteLine( formatter, "index", "array elements", "char"
```

```
);
        Console.WriteLine( formatter, "-----", "--------------", "----"
);

        // Convert byte array elements to char values.
        BAToChar( byteArray, 0 );
        BAToChar( byteArray, 1 );
        BAToChar( byteArray, 3 );
        BAToChar( byteArray, 5 );
        BAToChar( byteArray, 7 );
        BAToChar( byteArray, 9 );
        BAToChar( byteArray, 11 );
        BAToChar( byteArray, 13 );
        BAToChar( byteArray, 15 );
    }
}

/*
This example of the BitConverter.ToChar(byte[ ], int)
method generates the following output. It converts
elements of a byte array to char values.

initial byte array
------------------
20-00-00-2A-00-41-00-7D-00-C5-00-A8-03-29-04-AC-20

index   array elements    char
-----   --------------    ----
    0            20-00
    1            00-00
    3            2A-00       *
    5            41-00       A
    7            7D-00       }
    9            C5-00       Å
   11            A8-03       Ψ
   13            29-04       Щ
   15            AC-20       €
*/
```

Remarks

The ToChar method converts the bytes from index `startIndex` to `startIndex` + 1 to a Char value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(ReadOnlySpan<Byte>)

Converts a read-only byte span into a character.

C#

```
public static char ToChar (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Char

A character representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than the length of a Char.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToDouble Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToDouble(ReadOnlySpan<Byte>)	Converts a read-only byte span into a double-precision floating-point value.
ToDouble(Byte[], Int32)	Returns a double-precision floating point number converted from eight bytes at a specified position in a byte array.

ToDouble(ReadOnlySpan<Byte>)

Converts a read-only byte span into a double-precision floating-point value.

C#

```
public static double ToDouble (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Double

A double-precision floating-point value that represents the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than the length of a Double value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToDouble(Byte[], Int32)

Returns a double-precision floating point number converted from eight bytes at a specified position in a byte array.

C#

```
public static double ToDouble (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes that includes the eight bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

Double

A double-precision floating point number formed by eight bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 7, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Double values with the `ToDouble` method.

C#

```
// Example of the BitConverter.ToDouble method.
using System;

class BytesToDoubleDemo
{
    const string formatter = "{0,5}{1,27}{2,27:E16}";

    // Convert eight byte array elements to a double and display it.
    public static void BAToDouble( byte[ ] bytes, int index )
    {
        double value = BitConverter.ToDouble( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 8 ), value );
    }

    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes )
    {
        const int rowSize = 20;
        int iter;

        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
    }

    public static void Main( )
```

```
    {
        byte[ ] byteArray = {
              0,   0,   0,   0,   0,   0,   0,   0, 240,  63,
              0,   0,   0,   0,   0, 224, 111,  64,   0,   0,
            224, 255, 255, 255, 239,  65,   0,   0,   0,   0,
              0,   0, 112,  63,   0,   0,   0,   0,   0,   0,
            240,  61, 223, 136,  30,  28, 254, 116, 170,   1,
            250,  89, 140,  66, 202, 192, 243,  63, 251,  89,
            140,  66, 202, 192, 243,  63, 252,  89, 140,  66,
            202, 192, 243,  63,  82, 211, 187, 188, 232, 126,
             61, 126, 255, 255, 255, 255, 255, 255, 239, 255,
            255, 255, 255, 255, 255, 239, 127,   1,   0,   0,
              0,   0,   0,   0,   0, 248, 255,   0,   0,   0,
              0,   0,   0, 240, 255,   0,   0,   0,   0,   0,
              0, 240, 127 };

        Console.WriteLine(
            "This example of the BitConverter.ToDouble( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to double values.\n" );

        WriteMultiLineByteArray( byteArray );

        Console.WriteLine( formatter, "index", "array elements",
            "double" );
        Console.WriteLine( formatter, "-----", "--------------",
            "------" );

        // Convert byte array elements to double values.
        BAToDouble( byteArray, 0 );
        BAToDouble( byteArray, 2 );
        BAToDouble( byteArray, 10 );
        BAToDouble( byteArray, 18 );
        BAToDouble( byteArray, 26 );
        BAToDouble( byteArray, 34 );
        BAToDouble( byteArray, 42 );
        BAToDouble( byteArray, 50 );
        BAToDouble( byteArray, 58 );
        BAToDouble( byteArray, 66 );
        BAToDouble( byteArray, 74 );
        BAToDouble( byteArray, 82 );
        BAToDouble( byteArray, 89 );
        BAToDouble( byteArray, 97 );
        BAToDouble( byteArray, 99 );
        BAToDouble( byteArray, 107 );
        BAToDouble( byteArray, 115 );
    }
}

/*
This example of the BitConverter.ToDouble( byte[ ], int )
method generates the following output. It converts elements
of a byte array to double values.

initial byte array
```

```
------------------
00-00-00-00-00-00-00-00-F0-3F-00-00-00-00-00-E0-6F-40-00-00-
E0-FF-FF-FF-EF-41-00-00-00-00-00-00-70-3F-00-00-00-00-00-00-
F0-3D-DF-88-1E-1C-FE-74-AA-01-FA-59-8C-42-CA-C0-F3-3F-FB-59-
8C-42-CA-C0-F3-3F-FC-59-8C-42-CA-C0-F3-3F-52-D3-BB-BC-E8-7E-
3D-7E-FF-FF-FF-FF-FF-FF-EF-FF-FF-FF-FF-FF-FF-EF-7F-01-00-00-
00-00-00-00-00-F8-FF-00-00-00-00-00-00-F0-FF-00-00-00-00-00-
00-F0-7F

index             array elements                     double
-----             --------------                     ------
    0    00-00-00-00-00-00-00-00    0.0000000000000000E+000
    2    00-00-00-00-00-00-F0-3F    1.0000000000000000E+000
   10    00-00-00-00-00-E0-6F-40    2.5500000000000000E+002
   18    00-00-E0-FF-FF-FF-EF-41    4.2949672950000000E+009
   26    00-00-00-00-00-00-70-3F    3.9062500000000000E-003
   34    00-00-00-00-00-00-F0-3D    2.3283064365386963E-010
   42    DF-88-1E-1C-FE-74-AA-01    1.2345678901234500E-300
   50    FA-59-8C-42-CA-C0-F3-3F    1.2345678901234565E+000
   58    FB-59-8C-42-CA-C0-F3-3F    1.2345678901234567E+000
   66    FC-59-8C-42-CA-C0-F3-3F    1.2345678901234569E+000
   74    52-D3-BB-BC-E8-7E-3D-7E    1.2345678901234569E+300
   82    FF-FF-FF-FF-FF-FF-EF-FF   -1.7976931348623157E+308
   89    FF-FF-FF-FF-FF-FF-EF-7F    1.7976931348623157E+308
   97    01-00-00-00-00-00-00-00    4.9406564584124654E-324
   99    00-00-00-00-00-00-F8-FF                        NaN
  107    00-00-00-00-00-00-F0-FF                  -Infinity
  115    00-00-00-00-00-00-F0-7F                   Infinity
*/
```

Remarks

The ToDouble method converts the bytes from index `startIndex` to `startIndex` + 7 to a Double value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToHalf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToHalf(ReadOnlySpan<Byte>)	Converts a read-only byte span into a half-precision floating-point value.
ToHalf(Byte[], Int32)	Returns a laft-precision floating point number converted from two bytes at a specified position in a byte array.

ToHalf(ReadOnlySpan<Byte>)

Converts a read-only byte span into a half-precision floating-point value.

C#

```
public static Half ToHalf (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Half

A half-precision floating-point value that represents the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 2.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToHalf(Byte[], Int32)

Returns a laft-precision floating point number converted from two bytes at a specified position in a byte array.

```
C#

public static Half ToHalf (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes that includes the two bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

Half

A half-precision floating point number formed by two bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` equals the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToInt16 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToInt16(Byte[], Int32)	Returns a 16-bit signed integer converted from two bytes at a specified position in a byte array.
ToInt16(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 16-bit signed integer.

ToInt16(Byte[], Int32)

Returns a 16-bit signed integer converted from two bytes at a specified position in a byte array.

C#

```
public static short ToInt16 (byte[] value, int startIndex);
```

Parameters

value Byte[]

An array of bytes that includes the two bytes to convert.

startIndex Int32

The starting position within `value`.

Returns

Int16

A 16-bit signed integer formed by two bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` equals the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Int16 values with the `ToInt16` method.

C#

```
// Example of the BitConverter.ToInt16 method.
using System;

class BytesToInt16Demo
{
    const string formatter = "{0,5}{1,17}{2,10}";

    // Convert two byte array elements to a short and display it.
    public static void BAToInt16( byte[ ] bytes, int index )
    {
        short value = BitConverter.ToInt16( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 2 ), value );
    }

    public static void Main( )
    {
        byte[ ] byteArray =
            { 15, 0, 0, 128, 16, 39, 240, 216, 241, 255, 127 };

        Console.WriteLine(
            "This example of the BitConverter.ToInt16( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to short values.\n" );
        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );
        Console.WriteLine( BitConverter.ToString( byteArray ) );
        Console.WriteLine( );
        Console.WriteLine( formatter, "index", "array elements", "short"
);
```

```
        Console.WriteLine( formatter, "-----", "--------------", "-----"
);

        // Convert byte array elements to short values.
        BAToInt16( byteArray, 1 );
        BAToInt16( byteArray, 0 );
        BAToInt16( byteArray, 8 );
        BAToInt16( byteArray, 4 );
        BAToInt16( byteArray, 6 );
        BAToInt16( byteArray, 9 );
        BAToInt16( byteArray, 2 );
    }
}

/*
This example of the BitConverter.ToInt16( byte[ ], int )
method generates the following output. It converts elements
of a byte array to short values.

initial byte array
------------------
0F-00-00-80-10-27-F0-D8-F1-FF-7F

index   array elements     short
-----   --------------     -----
    1            00-00         0
    0            0F-00        15
    8            F1-FF       -15
    4            10-27     10000
    6            F0-D8    -10000
    9            FF-7F     32767
    2            00-80    -32768
*/
```

Remarks

The ToInt16 method converts the bytes from index `startIndex` to `startIndex` + 1 to an Int16 value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(Int16)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(ReadOnlySpan<Byte>)

Converts a read-only byte span into a 16-bit signed integer.

C#

```
public static short ToInt16 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Int16

A 16-bit signed integer representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 2.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.ToInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToInt32(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 32-bit signed integer.
ToInt32(Byte[], Int32)	Returns a 32-bit signed integer converted from four bytes at a specified position in a byte array.

ToInt32(ReadOnlySpan<Byte>)

Converts a read-only byte span into a 32-bit signed integer.

C#

```
public static int ToInt32 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Int32

A 32-bit signed integer representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 4.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToInt32(Byte[], Int32)

Returns a 32-bit signed integer converted from four bytes at a specified position in a byte array.

C#

```
public static int ToInt32 (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes that includes the four bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

Int32

A 32-bit signed integer formed by four bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 3, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following example uses the ToInt32 method to create Int32 values from a four-byte array and from the upper four bytes of an eight-byte array. It also uses the GetBytes(Int32) and ToInt32 methods to round-trip an Int32 value.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Create an Integer from a 4-byte array.
      Byte[] bytes1 = { 0xEC, 0x00, 0x00, 0x00 };
      Console.WriteLine("{0}--> 0x{1:X4} ({1:N0})", FormatBytes(bytes1),
                                     BitConverter.ToInt32(bytes1, 0));
      // Create an Integer from the upper four bytes of a byte array.
      Byte[] bytes2 = BitConverter.GetBytes(Int64.MaxValue / 2);
      Console.WriteLine("{0}--> 0x{1:X4} ({1:N0})", FormatBytes(bytes2),
                                     BitConverter.ToInt32(bytes2, 4));

      // Round-trip an integer value.
      int original = (int) Math.Pow(16, 3);
      Byte[] bytes3 = BitConverter.GetBytes(original);
      int restored = BitConverter.ToInt32(bytes3, 0);
      Console.WriteLine("0x{0:X4} ({0:N0}) --> {1} --> 0x{2:X4}
({2:N0})", original,
                        FormatBytes(bytes3), restored);
   }

   private static string FormatBytes(Byte[] bytes)
   {
       string value = "";
       foreach (var byt in bytes)
          value += string.Format("{0:X2} ", byt);

       return value;
   }
}
// The example displays the following output:
//       EC 00 00 00 --> 0x00EC (236)
```

```
//       FF FF FF FF FF FF FF 3F --> 0x3FFFFFFF (1,073,741,823)
//       0x1000 (4,096) --> 00 10 00 00  --> 0x1000 (4,096)
```

Remarks

The ToInt32 method converts the bytes from index `startIndex` to `startIndex` + 3 to an Int32 value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of BitConverter.

See also

- GetBytes(Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToInt64(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 64-bit signed integer.
ToInt64(Byte[], Int32)	Returns a 64-bit signed integer converted from eight bytes at a specified position in a byte array.

ToInt64(ReadOnlySpan<Byte>)

Converts a read-only byte span into a 64-bit signed integer.

C#

```
public static long ToInt64 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Int64

A 64-bit signed integer representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 8.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToInt64(Byte[], Int32)

Returns a 64-bit signed integer converted from eight bytes at a specified position in a byte array.

C#

```
public static long ToInt64 (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes that includes the eight bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

Int64

A 64-bit signed integer formed by eight bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 7, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Int64 values with the `ToInt64` method.

C#

```
// Example of the BitConverter.ToInt64 method.
using System;

class BytesToInt64Demo
{
    const string formatter = "{0,5}{1,27}{2,24}";

    // Convert eight byte array elements to a long and display it.
    public static void BAToInt64( byte[ ] bytes, int index )
    {
        long value = BitConverter.ToInt64( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 8 ), value );
    }

    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes )
    {
        const int rowSize = 20;
        int iter;

        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
    }

    public static void Main( )
```

```
    {
        byte[ ] byteArray = {
              0,  54, 101, 196, 255, 255, 255, 255,   0,   0,
              0,   0,   0,   0,   0,   0, 128,   0, 202, 154,
             59,   0,   0,   0,   0,   1,   0,   0,   0,   0,
            255, 255, 255, 255,   1,   0,   0, 255, 255, 255,
            255, 255, 255, 255, 127,  86,  85,  85,  85,  85,
             85, 255, 255, 170, 170, 170, 170, 170, 170,   0,
              0, 100, 167, 179, 182, 224,  13,   0,   0, 156,
             88,  76,  73,  31, 242 };

        Console.WriteLine(
            "This example of the BitConverter.ToInt64( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to long values.\r\n" );

        WriteMultiLineByteArray( byteArray );

        Console.WriteLine( formatter, "index", "array elements", "long"
);
        Console.WriteLine( formatter, "-----", "--------------", "----"
);

        // Convert byte array elements to long values.
        BAToInt64( byteArray, 8 );
        BAToInt64( byteArray, 5 );
        BAToInt64( byteArray, 34 );
        BAToInt64( byteArray, 17 );
        BAToInt64( byteArray, 0 );
        BAToInt64( byteArray, 21 );
        BAToInt64( byteArray, 26 );
        BAToInt64( byteArray, 53 );
        BAToInt64( byteArray, 45 );
        BAToInt64( byteArray, 59 );
        BAToInt64( byteArray, 67 );
        BAToInt64( byteArray, 37 );
        BAToInt64( byteArray, 9 );
    }
}

/*
This example of the BitConverter.ToInt64( byte[ ], int )
method generates the following output. It converts elements
of a byte array to long values.

initial byte array
------------------
00-36-65-C4-FF-FF-FF-FF-00-00-00-00-00-00-00-00-80-00-CA-9A-
3B-00-00-00-00-01-00-00-00-00-FF-FF-FF-FF-01-00-00-FF-FF-FF-
FF-FF-FF-FF-7F-56-55-55-55-55-55-FF-FF-AA-AA-AA-AA-AA-AA-00-
00-64-A7-B3-B6-E0-0D-00-00-9C-58-4C-49-1F-F2

index             array elements                    long
-----             --------------                    ----
    8    00-00-00-00-00-00-00-00                       0
```

```
     5    FF-FF-FF-00-00-00-00-00                16777215
    34    01-00-00-FF-FF-FF-FF-FF               -16777215
    17    00-CA-9A-3B-00-00-00-00              1000000000
     0    00-36-65-C4-FF-FF-FF-FF             -1000000000
    21    00-00-00-00-01-00-00-00              4294967296
    26    00-00-00-00-FF-FF-FF-FF             -4294967296
    53    AA-AA-AA-AA-AA-AA-00-00         187649984473770
    45    56-55-55-55-55-55-FF-FF        -187649984473770
    59    00-00-64-A7-B3-B6-E0-0D     1000000000000000000
    67    00-00-9C-58-4C-49-1F-F2    -1000000000000000000
    37    FF-FF-FF-FF-FF-FF-FF-7F     9223372036854775807
     9    00-00-00-00-00-00-00-80    -9223372036854775808
*/
```

Remarks

The ToInt64 method converts the bytes from index `startIndex` to `startIndex` + 7 to a Int64 value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(Int64)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.ToSingle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToSingle(ReadOnlySpan<Byte>)	Converts a read-only byte span into a single-precision floating-point value.
ToSingle(Byte[], Int32)	Returns a single-precision floating point number converted from four bytes at a specified position in a byte array.

ToSingle(ReadOnlySpan<Byte>)

Converts a read-only byte span into a single-precision floating-point value.

C#

```
public static float ToSingle (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

Single

A single-precision floating-point value representing the converted bytes.

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than the length of a Single value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToSingle(Byte[], Int32)

Returns a single-precision floating point number converted from four bytes at a specified position in a byte array.

C#

```
public static float ToSingle (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes.

`startIndex` Int32

The starting position within `value`.

Returns

Single

A single-precision floating point number formed by four bytes beginning at `startIndex`.

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 3, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to Single values with the `ToSingle` method.

C#

```
// Example of the BitConverter.ToSingle method.
using System;

class BytesToSingleDemo
{
    const string formatter = "{0,5}{1,17}{2,18:E7}";

    // Convert four byte array elements to a float and display it.
    public static void BAToSingle( byte[ ] bytes, int index )
    {
        float value = BitConverter.ToSingle( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 4 ), value );
    }

    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes )
    {
        const int rowSize = 20;
        int iter;

        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
    }

    public static void Main( )
```

```
    {
        byte[ ] byteArray = {
              0,   0,   0,   0, 128,  63,   0,   0, 112,  65,
              0, 255, 127,  71,   0,   0, 128,  59,   0,   0,
            128,  47,  73,  70, 131,   5,  75,   6, 158,  63,
             77,   6, 158,  63,  80,   6, 158,  63,  30,  55,
            190, 121, 255, 255, 127, 255, 255, 127, 127,   1,
              0,   0,   0, 192, 255,   0,   0, 128, 255,   0,
              0, 128, 127 };

        Console.WriteLine(
            "This example of the BitConverter.ToSingle( byte( ), " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to float values.\n" );

        WriteMultiLineByteArray( byteArray );

        Console.WriteLine( formatter, "index", "array elements",
            "float" );
        Console.WriteLine( formatter, "-----", "--------------",
            "-----" );

        // Convert byte array elements to float values.
        BAToSingle( byteArray, 0 );
        BAToSingle( byteArray, 2 );
        BAToSingle( byteArray, 6 );
        BAToSingle( byteArray, 10 );
        BAToSingle( byteArray, 14 );
        BAToSingle( byteArray, 18 );
        BAToSingle( byteArray, 22 );
        BAToSingle( byteArray, 26 );
        BAToSingle( byteArray, 30 );
        BAToSingle( byteArray, 34 );
        BAToSingle( byteArray, 38 );
        BAToSingle( byteArray, 42 );
        BAToSingle( byteArray, 45 );
        BAToSingle( byteArray, 49 );
        BAToSingle( byteArray, 51 );
        BAToSingle( byteArray, 55 );
        BAToSingle( byteArray, 59 );
    }
}

/*
This example of the BitConverter.ToSingle( byte( ), int )
method generates the following output. It converts elements
of a byte array to float values.

initial byte array
------------------
00-00-00-00-80-3F-00-00-70-41-00-FF-7F-47-00-00-80-3B-00-00-
80-2F-49-46-83-05-4B-06-9E-3F-4D-06-9E-3F-50-06-9E-3F-1E-37-
BE-79-FF-FF-7F-FF-FF-7F-7F-01-00-00-00-C0-FF-00-00-80-FF-00-
00-80-7F
```

```
 index   array elements             float
 -----   --------------             -----
     0      00-00-00-00    0.0000000E+000
     2      00-00-80-3F    1.0000000E+000
     6      00-00-70-41    1.5000000E+001
    10      00-FF-7F-47    6.5535000E+004
    14      00-00-80-3B    3.9062500E-003
    18      00-00-80-2F    2.3283064E-010
    22      49-46-83-05    1.2345000E-035
    26      4B-06-9E-3F    1.2345671E+000
    30      4D-06-9E-3F    1.2345673E+000
    34      50-06-9E-3F    1.2345676E+000
    38      1E-37-BE-79    1.2345679E+035
    42      FF-FF-7F-FF   -3.4028235E+038
    45      FF-FF-7F-7F    3.4028235E+038
    49      01-00-00-00    1.4012985E-045
    51      00-00-C0-FF               NaN
    55      00-00-80-FF         -Infinity
    59      00-00-80-7F          Infinity
 */
```

Remarks

The ToSingle method converts the bytes from index `startIndex` to `startIndex` + 3 to a Single value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of BitConverter.

See also

- GetBytes(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of each element of a specified array of bytes to its equivalent hexadecimal string representation.

Overloads

Expand table

ToString(Byte[])	Converts the numeric value of each element of a specified array of bytes to its equivalent hexadecimal string representation.
ToString(Byte[], Int32)	Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.
ToString(Byte[], Int32, Int32)	Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.

ToString(Byte[])

Converts the numeric value of each element of a specified array of bytes to its equivalent hexadecimal string representation.

C#

```
public static string ToString (byte[] value);
```

Parameters

`value` Byte[]

An array of bytes.

Returns

String

A string of hexadecimal pairs separated by hyphens, where each pair represents the corresponding element in `value`; for example, "7F-2C-4A-00".

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following code example converts Byte arrays to String objects with the `ToString` method.

C#

```
// Example of the BitConverter.ToString( byte[ ] ) method.
using System;

class BytesToStringDemo
{
    // Display a byte array with a name.
    public static void WriteByteArray( byte[ ] bytes, string name )
    {
        const string underLine = "--------------------------------";

        Console.WriteLine( name );
        Console.WriteLine( underLine.Substring( 0,
            Math.Min( name.Length, underLine.Length ) ) );
        Console.WriteLine( BitConverter.ToString( bytes ) );
        Console.WriteLine( );
    }

    public static void Main( )
    {
        byte[ ] arrayOne = {
             0,   1,   2,   4,   8,  16,  32,  64, 128, 255 };

        byte[ ] arrayTwo = {
            32,   0,   0,  42,   0,  65,   0, 125,   0, 197,
             0, 168,   3,  41,   4, 172,  32 };

        byte[ ] arrayThree = {
            15,   0,   0, 128,  16,  39, 240, 216, 241, 255,
           127 };

        byte[ ] arrayFour = {
            15,   0,   0,   0,   0,  16,   0, 255,   3,   0,
             0, 202, 154,  59, 255, 255, 255, 255, 127 };
```

```
        Console.WriteLine( "This example of the " +
            "BitConverter.ToString( byte[ ] ) \n" +
            "method generates the following output.\n" );

        WriteByteArray( arrayOne, "arrayOne" );
        WriteByteArray( arrayTwo, "arrayTwo" );
        WriteByteArray( arrayThree, "arrayThree" );
        WriteByteArray( arrayFour, "arrayFour" );
    }
}

/*
This example of the BitConverter.ToString( byte[ ] )
method generates the following output.

arrayOne
--------
00-01-02-04-08-10-20-40-80-FF

arrayTwo
--------
20-00-00-2A-00-41-00-7D-00-C5-00-A8-03-29-04-AC-20

arrayThree
----------
0F-00-00-80-10-27-F0-D8-F1-FF-7F

arrayFour
---------
0F-00-00-00-00-10-00-FF-03-00-00-CA-9A-3B-FF-FF-FF-FF-7F
*/
```

Remarks

All the elements of `value` are converted.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString(Byte[], Int32)

Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.

```
C#
public static string ToString (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes.

`startIndex` Int32

The starting position within `value`.

Returns

String

A string of hexadecimal pairs separated by hyphens, where each pair represents the corresponding element in a subarray of `value`; for example, "7F-2C-4A-00".

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts the part of a Byte array starting at the specified `startIndex` to a String with the `ToString` method.

C#

```
// Example of some BitConverter.ToString( ) method overloads.
using System;

class BytesToStringDemo
{
    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes,
        string name )
    {
        const int rowSize = 20;
        const string underLine = "--------------------------------";
        int iter;

        Console.WriteLine( name );
        Console.WriteLine( underLine.Substring( 0,
            Math.Min( name.Length, underLine.Length ) ) );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
    }

    public static void Main( )
    {
        byte[ ] arrayOne = {
              0,   0,   0,   0, 128,  63,   0,   0, 112,  65,
              0, 255, 127,  71,   0,   0, 128,  59,   0,   0,
            128,  47,  73,  70, 131,   5,  75,   6, 158,  63,
             77,   6, 158,  63,  80,   6, 158,  63,  30,  55,
            190, 121, 255, 255, 127, 255, 255, 127, 127,   1,
              0,   0,   0, 192, 255,   0,   0, 128, 255,   0,
              0, 128, 127 };

        byte[ ] arrayTwo = {
            255, 255, 255,   0,   0,  20,   0,  33,   0,   0,
              0,   1,   0,   0,   0, 100, 167, 179, 182, 224,
             13,   0, 202, 154,  59,   0, 143,  91,   0, 170,
            170, 170, 170, 170, 170,   0,   0, 232, 137,   4,
             35, 199, 138, 255, 232, 244, 255, 252, 205, 255,
            255, 129 };

        byte[ ] arrayThree = {
              0, 222,   0,   0,   0, 224, 111,  64,   0,   0,
            224, 255, 255, 255, 239,  65,   0,   0, 131,   0,
              0,   0, 112,  63,   0, 143,   0, 100,   0,   0,
            240,  61, 223, 136,  30,  28, 254, 116, 170,   1,
```

```
            250,  89, 140,  66, 202, 192, 243,  63, 251,  89,
            140,  66, 202, 192, 243,  63, 252,  89, 140,  66,
            202, 192, 243,  63,  82, 211, 187, 188, 232, 126,
            255, 255, 255, 244, 255, 239, 127,   1,   0,   0,
              0,  10,  17,   0,   0, 248, 255,   0,  88,   0,
             91,   0,   0, 240, 255,   0,   0, 240, 157 };

        Console.WriteLine( "This example of the\n" +
            "  BitConverter.ToString( byte[ ], int ) and \n" +
            "  BitConverter.ToString( byte[ ], int, int ) \n" +
            "methods generates the following output.\n" );

        WriteMultiLineByteArray( arrayOne, "arrayOne" );
        WriteMultiLineByteArray( arrayTwo, "arrayTwo" );
        WriteMultiLineByteArray( arrayThree, "arrayThree" );
    }
}

/*
This example of the
  BitConverter.ToString( byte[ ], int ) and
  BitConverter.ToString( byte[ ], int, int )
methods generates the following output.

arrayOne
--------
00-00-00-00-80-3F-00-00-70-41-00-FF-7F-47-00-00-80-3B-00-00-
80-2F-49-46-83-05-4B-06-9E-3F-4D-06-9E-3F-50-06-9E-3F-1E-37-
BE-79-FF-FF-7F-FF-FF-7F-7F-01-00-00-00-C0-FF-00-00-80-FF-00-
00-80-7F

arrayTwo
--------
FF-FF-FF-00-00-14-00-21-00-00-00-01-00-00-00-64-A7-B3-B6-E0-
0D-00-CA-9A-3B-00-8F-5B-00-AA-AA-AA-AA-AA-AA-00-00-E8-89-04-
23-C7-8A-FF-E8-F4-FF-FC-CD-FF-FF-81

arrayThree
----------
00-DE-00-00-00-E0-6F-40-00-00-E0-FF-FF-FF-EF-41-00-00-83-00-
00-00-70-3F-00-8F-00-64-00-00-F0-3D-DF-88-1E-1C-FE-74-AA-01-
FA-59-8C-42-CA-C0-F3-3F-FB-59-8C-42-CA-C0-F3-3F-FC-59-8C-42-
CA-C0-F3-3F-52-D3-BB-BC-E8-7E-FF-FF-FF-F4-FF-EF-7F-01-00-00-
00-0A-11-00-00-F8-FF-00-58-00-5B-00-00-F0-FF-00-00-F0-9D
*/
```

Remarks

The elements from array position `startIndex` to the end of the array are converted.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Byte[], Int32, Int32)

Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.

C#

```
public static string ToString (byte[] value, int startIndex, int length);
```

Parameters

`value` Byte[]

An array of bytes that includes the bytes to convert.

`startIndex` Int32

The starting position within `value`.

`length` Int32

The number of array elements in `value` to convert.

Returns

String

A string of hexadecimal pairs separated by hyphens, where each pair represents the corresponding element in a subarray of `value`; for example, "7F-2C-4A-00".

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero.

-or-

`startIndex` is greater than zero and is greater than or equal to the length of `value`.

ArgumentException

The combination of `startIndex` and `length` does not specify a position within `value`; that is, the `startIndex` parameter is greater than the length of `value` minus the `length` parameter.

Examples

The following example uses the ToString method to convert part of a byte array, starting at the specified `startIndex` and with the specified `length`, to a string.

C#

```
// Example of some BitConverter.ToString( ) method overloads.
using System;

class BytesToStringDemo
{
    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes,
        string name )
    {
        const int rowSize = 20;
        const string underLine = "--------------------------------";
        int iter;

        Console.WriteLine( name );
        Console.WriteLine( underLine.Substring( 0,
            Math.Min( name.Length, underLine.Length ) ) );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
```

```
    }

    public static void Main( )
    {
        byte[ ] arrayOne = {
              0,   0,   0,   0, 128,  63,   0,   0, 112,  65,
              0, 255, 127,  71,   0,   0, 128,  59,   0,   0,
            128,  47,  73,  70, 131,   5,  75,   6, 158,  63,
             77,   6, 158,  63,  80,   6, 158,  63,  30,  55,
            190, 121, 255, 255, 127, 255, 255, 127, 127,   1,
              0,   0,   0, 192, 255,   0,   0, 128, 255,   0,
              0, 128, 127 };

        byte[ ] arrayTwo = {
            255, 255, 255,   0,   0,  20,   0,  33,   0,   0,
              0,   1,   0,   0,   0, 100, 167, 179, 182, 224,
             13,   0, 202, 154,  59,   0, 143,  91,   0, 170,
            170, 170, 170, 170, 170,   0,   0, 232, 137,   4,
             35, 199, 138, 255, 232, 244, 255, 252, 205, 255,
            255, 129 };

        byte[ ] arrayThree = {
              0, 222,   0,   0,   0, 224, 111,  64,   0,   0,
            224, 255, 255, 255, 239,  65,   0,   0, 131,   0,
              0,   0, 112,  63,   0, 143,   0, 100,   0,   0,
            240,  61, 223, 136,  30,  28, 254, 116, 170,   1,
            250,  89, 140,  66, 202, 192, 243,  63, 251,  89,
            140,  66, 202, 192, 243,  63, 252,  89, 140,  66,
            202, 192, 243,  63,  82, 211, 187, 188, 232, 126,
            255, 255, 255, 244, 255, 239, 127,   1,   0,   0,
              0,  10,  17,   0,   0, 248, 255,   0,  88,   0,
             91,   0,   0, 240, 255,   0,   0, 240, 157 };

        Console.WriteLine( "This example of the\n" +
            "  BitConverter.ToString( byte[ ], int ) and \n" +
            "  BitConverter.ToString( byte[ ], int, int ) \n" +
            "methods generates the following output.\n" );

        WriteMultiLineByteArray( arrayOne, "arrayOne" );
        WriteMultiLineByteArray( arrayTwo, "arrayTwo" );
        WriteMultiLineByteArray( arrayThree, "arrayThree" );
    }
}

/*
This example of the
  BitConverter.ToString( byte[ ], int ) and
  BitConverter.ToString( byte[ ], int, int )
methods generates the following output.

arrayOne
--------
00-00-00-00-80-3F-00-00-70-41-00-FF-7F-47-00-00-80-3B-00-00-
80-2F-49-46-83-05-4B-06-9E-3F-4D-06-9E-3F-50-06-9E-3F-1E-37-
BE-79-FF-FF-7F-FF-FF-7F-7F-01-00-00-00-C0-FF-00-00-80-FF-00-
```

```
00-80-7F

arrayTwo
--------
FF-FF-FF-00-00-14-00-21-00-00-00-01-00-00-00-64-A7-B3-B6-E0-
0D-00-CA-9A-3B-00-8F-5B-00-AA-AA-AA-AA-AA-AA-00-00-E8-89-04-
23-C7-8A-FF-E8-F4-FF-FC-CD-FF-FF-81

arrayThree
----------
00-DE-00-00-00-E0-6F-40-00-00-E0-FF-FF-FF-EF-41-00-00-83-00-
00-00-70-3F-00-8F-00-64-00-00-F0-3D-DF-88-1E-1C-FE-74-AA-01-
FA-59-8C-42-CA-C0-F3-3F-FB-59-8C-42-CA-C0-F3-3F-FC-59-8C-42-
CA-C0-F3-3F-52-D3-BB-BC-E8-7E-FF-FF-FF-F4-FF-EF-7F-01-00-00-
00-0A-11-00-00-F8-FF-00-58-00-5B-00-00-F0-FF-00-00-F0-9D
*/
```

Remarks

The `length` elements from array position `startIndex` are converted. If `length` equals zero, the method returns String.Empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

BitConverter.ToUInt16 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToUInt16(ReadOnlySpan<Byte>)	Converts a read-only byte-span into a 16-bit unsigned integer.
ToUInt16(Byte[], Int32)	Returns a 16-bit unsigned integer converted from two bytes at a specified position in a byte array.

ToUInt16(ReadOnlySpan<Byte>)

> Important
>
> This API is not CLS-compliant.

Converts a read-only byte-span into a 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

UInt16

An 16-bit unsigned integer representing the converted bytes.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 2.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToUInt16(Byte[], Int32)

> Important
>
> This API is not CLS-compliant.

Returns a 16-bit unsigned integer converted from two bytes at a specified position in a byte array.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

The array of bytes that includes the two bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

UInt16

A 16-bit unsigned integer formed by two bytes beginning at `startIndex`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`startIndex` equals the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to UInt16 values with the `ToUInt16` method.

C#

```
// Example of the BitConverter.ToUInt16 method.
using System;

class BytesToUInt16Demo
{
    const string formatter = "{0,5}{1,17}{2,10}";

    // Convert two byte array elements to a ushort and display it.
    public static void BAToUInt16( byte[ ] bytes, int index )
    {
        ushort value = BitConverter.ToUInt16( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 2 ), value );
    }

    public static void Main( )
    {
        byte[] byteArray = {
            15, 0, 0, 255, 3, 16, 39, 255, 255, 127 };
```

```
        Console.WriteLine(
            "This example of the BitConverter.ToUInt16( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to ushort values.\n" );
        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );
        Console.WriteLine( BitConverter.ToString( byteArray ) );
        Console.WriteLine( );
        Console.WriteLine( formatter, "index", "array elements",
            "ushort" );
        Console.WriteLine( formatter, "-----", "--------------",
            "------" );

        // Convert byte array elements to ushort values.
        BAToUInt16( byteArray, 1 );
        BAToUInt16( byteArray, 0 );
        BAToUInt16( byteArray, 3 );
        BAToUInt16( byteArray, 5 );
        BAToUInt16( byteArray, 8 );
        BAToUInt16( byteArray, 7 );
    }
}

/*
This example of the BitConverter.ToUInt16( byte[ ], int )
method generates the following output. It converts elements
of a byte array to ushort values.

initial byte array
------------------
0F-00-00-FF-03-10-27-FF-FF-7F

index   array elements    ushort
-----   --------------    ------
    1            00-00         0
    0            0F-00        15
    3            FF-03      1023
    5            10-27     10000
    8            FF-7F     32767
    7            FF-FF     65535
*/
```

Remarks

The ToUInt16 method converts the bytes from index `startIndex` to `startIndex` + 1 to a UInt16 value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(UInt16)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToUInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToUInt32(ReadOnlySpan<Byte>)	Converts a read-only byte span into a 32-bit unsigned integer.
ToUInt32(Byte[], Int32)	Returns a 32-bit unsigned integer converted from four bytes at a specified position in a byte array.

ToUInt32(ReadOnlySpan<Byte>)

> Important
>
> This API is not CLS-compliant.

Converts a read-only byte span into a 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

UInt32

A 32-bit unsigned integer representing the converted bytes.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 4.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToUInt32(Byte[], Int32)

> Important
>
> This API is not CLS-compliant.

Returns a 32-bit unsigned integer converted from four bytes at a specified position in a byte array.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes.

`startIndex` Int32

The starting position within `value`.

Returns

UInt32

A 32-bit unsigned integer formed by four bytes beginning at `startIndex`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 3, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to UInt32 values with the `ToUInt32` method.

C#

```
// Example of the BitConverter.ToUInt32 method.
using System;

class BytesToUInt32Demo
{
    const string formatter = "{0,5}{1,17}{2,15}";

    // Convert four byte array elements to a uint and display it.
    public static void BAToUInt32( byte[ ] bytes, int index )
    {
        uint value = BitConverter.ToUInt32( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 4 ), value );
    }

    public static void Main( )
    {
        byte[ ] byteArray = {
             15,   0,   0,   0,   0,  16,   0, 255,   3,   0,
```

```
                0, 202, 154,  59, 255, 255, 255, 255, 127 };

        Console.WriteLine(
            "This example of the BitConverter.ToUInt32( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to uint values.\n" );
        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );
        Console.WriteLine( BitConverter.ToString( byteArray ) );
        Console.WriteLine( );
        Console.WriteLine( formatter, "index", "array elements",
            "uint" );
        Console.WriteLine( formatter, "-----", "--------------",
            "----" );

        // Convert byte array elements to uint values.
        BAToUInt32( byteArray, 1 );
        BAToUInt32( byteArray, 0 );
        BAToUInt32( byteArray, 7 );
        BAToUInt32( byteArray, 3 );
        BAToUInt32( byteArray, 10 );
        BAToUInt32( byteArray, 15 );
        BAToUInt32( byteArray, 14 );
    }
}

/*
This example of the BitConverter.ToUInt32( byte[ ], int )
method generates the following output. It converts elements
of a byte array to uint values.

initial byte array
------------------
0F-00-00-00-00-10-00-FF-03-00-00-CA-9A-3B-FF-FF-FF-FF-7F

index   array elements           uint
-----   --------------           ----
    1      00-00-00-00              0
    0      0F-00-00-00             15
    7      FF-03-00-00           1023
    3      00-00-10-00        1048576
   10      00-CA-9A-3B     1000000000
   15      FF-FF-FF-7F     2147483647
   14      FF-FF-FF-FF     4294967295
*/
```

Remarks

The ToUInt32 method converts the bytes from index `startIndex` to `startIndex` + 3 to a UInt32 value. The order of bytes in the array must reflect the endianness of the

computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(UInt32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.ToUInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToUInt64(Byte[], Int32)	Returns a 64-bit unsigned integer converted from eight bytes at a specified position in a byte array.
ToUInt64(ReadOnlySpan<Byte>)	Converts bytes into an unsigned long.

ToUInt64(Byte[], Int32)

> Important
>
> This API is not CLS-compliant.

Returns a 64-bit unsigned integer converted from eight bytes at a specified position in a byte array.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (byte[] value, int startIndex);
```

Parameters

`value` Byte[]

An array of bytes that includes the eight bytes to convert.

`startIndex` Int32

The starting position within `value`.

Returns

UInt64

A 64-bit unsigned integer formed by the eight bytes beginning at `startIndex`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`startIndex` is greater than or equal to the length of `value` minus 7, and is less than or equal to the length of `value` minus 1.

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of `value` minus 1.

Examples

The following code example converts elements of Byte arrays to UInt64 values with the `ToUInt64` method.

C#

```
// Example of the BitConverter.ToUInt64 method.
using System;

class BytesToUInt64Demo
{
    const string formatter = "{0,5}{1,27}{2,24}";

    // Convert eight byte array elements to a ulong and display it.
    public static void BAToUInt64( byte[ ] bytes, int index )
    {
        ulong value = BitConverter.ToUInt64( bytes, index );

        Console.WriteLine( formatter, index,
            BitConverter.ToString( bytes, index, 8 ), value );
    }

    // Display a byte array, using multiple lines if necessary.
    public static void WriteMultiLineByteArray( byte[ ] bytes )
    {
        const int rowSize = 20;
```

```
        int iter;

        Console.WriteLine( "initial byte array" );
        Console.WriteLine( "------------------" );

        for( iter = 0; iter < bytes.Length - rowSize; iter += rowSize )
        {
            Console.Write(
                BitConverter.ToString( bytes, iter, rowSize ) );
            Console.WriteLine( "-" );
        }

        Console.WriteLine( BitConverter.ToString( bytes, iter ) );
        Console.WriteLine( );
    }

    public static void Main( )
    {
        byte[ ] byteArray = {
            255, 255, 255,   0,   0,   0,   0,   0,   0,   0,
              0,   1,   0,   0,   0, 100, 167, 179, 182, 224,
             13,   0, 202, 154,  59,   0,   0,   0,   0, 170,
            170, 170, 170, 170, 170,   0,   0, 232, 137,   4,
             35, 199, 138, 255, 255, 255, 255, 255, 255, 255,
            255, 127 };

        Console.WriteLine(
            "This example of the BitConverter.ToUInt64( byte[ ], " +
            "int ) \nmethod generates the following output. It " +
            "converts elements \nof a byte array to ulong values.\n" );

        WriteMultiLineByteArray( byteArray );

        Console.WriteLine( formatter, "index", "array elements",
            "ulong" );
        Console.WriteLine( formatter, "-----", "--------------",
            "------" );

        // Convert byte array elements to ulong values.
        BAToUInt64( byteArray, 3 );
        BAToUInt64( byteArray, 0 );
        BAToUInt64( byteArray, 21 );
        BAToUInt64( byteArray, 7 );
        BAToUInt64( byteArray, 29 );
        BAToUInt64( byteArray, 13 );
        BAToUInt64( byteArray, 35 );
        BAToUInt64( byteArray, 44 );
        BAToUInt64( byteArray, 43 );
    }
}

/*
This example of the BitConverter.ToUInt64( byte[ ], int )
method generates the following output. It converts elements
of a byte array to ulong values.
```

```
initial byte array
------------------
FF-FF-FF-00-00-00-00-00-00-00-00-01-00-00-00-64-A7-B3-B6-E0-
0D-00-CA-9A-3B-00-00-00-00-AA-AA-AA-AA-AA-AA-00-00-E8-89-04-
23-C7-8A-FF-FF-FF-FF-FF-FF-FF-FF-7F

index             array elements                   ulong
-----             --------------                  ------
    3    00-00-00-00-00-00-00-00                       0
    0    FF-FF-FF-00-00-00-00-00                16777215
   21    00-CA-9A-3B-00-00-00-00              1000000000
    7    00-00-00-00-01-00-00-00              4294967296
   29    AA-AA-AA-AA-AA-AA-00-00         187649984473770
   13    00-00-64-A7-B3-B6-E0-0D     1000000000000000000
   35    00-00-E8-89-04-23-C7-8A    10000000000000000000
   44    FF-FF-FF-FF-FF-FF-FF-7F     9223372036854775807
   43    FF-FF-FF-FF-FF-FF-FF-FF    18446744073709551615
*/
```

Remarks

The ToUInt64 method converts the bytes from index `startIndex` to `startIndex` + 7 to a UInt64 value. The order of bytes in the array must reflect the endianness of the computer system's architecture. For more information, see the Remarks section of the BitConverter class topic.

See also

- GetBytes(UInt64)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(ReadOnlySpan<Byte>)

> Important
>
> This API is not CLS-compliant.

Converts bytes into an unsigned long.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

A read-only span containing the bytes to convert.

Returns

UInt64

A 64-bit unsigned integer representing the converted bytes.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

The length of `value` is less than 8.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.TryWriteBytes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryWriteBytes(Span<Byte>, Int64)	Converts a 64-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Boolean)	Converts a Boolean into a span of bytes.
TryWriteBytes(Span<Byte>, Char)	Converts a character into a span of bytes.
TryWriteBytes(Span<Byte>, Double)	Converts a double-precision floating-point value into a span of bytes.
TryWriteBytes(Span<Byte>, Half)	Converts a half-precision floating-point value into a span of bytes.
TryWriteBytes(Span<Byte>, Int16)	Converts a 16-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Int32)	Converts a 32-bit signed integer into a span of bytes.
TryWriteBytes(Span<Byte>, Single)	Converts a single-precision floating-point value into a span of bytes.
TryWriteBytes(Span<Byte>, UInt16)	Converts an unsigned 16-bit integer into a span of bytes.
TryWriteBytes(Span<Byte>, UInt32)	Converts a 32-bit unsigned integer into a span of bytes.
TryWriteBytes(Span<Byte>, UInt64)	Converts an unsigned 64-bit integer into a span of bytes.

TryWriteBytes(Span<Byte>, Int64)

Converts a 64-bit signed integer into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, long value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted 64-bit signed integer.

`value` Int64

The 64-bit signed integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Boolean)

Converts a Boolean into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, bool value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted Boolean.

`value` Boolean

The Boolean to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Char)

Converts a character into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, char value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted character.

`value` Char

The character to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Double)

Converts a double-precision floating-point value into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, double value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted double-precision floating-point value.

`value` Double

The double-precision floating-point value to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Half)

Converts a half-precision floating-point value into a span of bytes.

```
C#
public static bool TryWriteBytes (Span<byte> destination, Half value);
```

Parameters

destination Span<Byte>

When this method returns, the bytes representing the converted half-precision floating-point value.

value Half

The half-precision floating-point value to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryWriteBytes(Span<Byte>, Int16)

Converts a 16-bit signed integer into a span of bytes.

```
C#
public static bool TryWriteBytes (Span<byte> destination, short value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted 16-bit signed integer.

`value` Int16

The 16-bit signed integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Int32)

Converts a 32-bit signed integer into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, int value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted 32-bit signed integer.

`value` Int32

The 32-bit signed integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Single)

Converts a single-precision floating-point value into a span of bytes.

C#

```
public static bool TryWriteBytes (Span<byte> destination, float value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted single-precision floating-point value.

`value` Single

The single-precision floating-point value to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, UInt16)

> Important
>
> This API is not CLS-compliant.

Converts an unsigned 16-bit integer into a span of bytes.

C#

```
[System.CLSCompliant(false)]
public static bool TryWriteBytes (Span<byte> destination, ushort value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted unsigned 16-bit integer.

`value` UInt16

The unsigned 16-bit integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, UInt32)

> Important
>
> This API is not CLS-compliant.

Converts a 32-bit unsigned integer into a span of bytes.

C#

```
[System.CLSCompliant(false)]
public static bool TryWriteBytes (Span<byte> destination, uint value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted unsigned 32-bit integer.

`value` UInt32

The unsigned 32-bit integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, UInt64)

> Important
>
> This API is not CLS-compliant.

Converts an unsigned 64-bit integer into a span of bytes.

C#

```
[System.CLSCompliant(false)]
public static bool TryWriteBytes (Span<byte> destination, ulong value);
```

Parameters

`destination` Span<Byte>

When this method returns, the bytes representing the converted unsigned 64-bit integer.

`value` UInt64

The unsigned 64-bit integer to convert.

Returns

Boolean

`true` if the conversion was successful; `false` otherwise.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

BitConverter.UInt16BitsToHalf(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified 16-bit unsigned integer to a half-precision floating point number.

C#

```
[System.CLSCompliant(false)]
public static Half UInt16BitsToHalf (ushort value);
```

Parameters

`value` UInt16

The number to convert.

Returns

Half

A half-precision floating point number whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.UInt32BitsToSingle(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified 32-bit unsigned integer to a single-precision floating point number.

C#

```
[System.CLSCompliant(false)]
public static float UInt32BitsToSingle (uint value);
```

Parameters

value UInt32

The number to convert.

Returns

Single

A single-precision floating point number whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

BitConverter.UInt64BitsToDouble(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified 64-bit unsigned integer to a double-precision floating point number.

C#

```
[System.CLSCompliant(false)]
public static double UInt64BitsToDouble (ulong value);
```

Parameters

`value` UInt64

The number to convert.

Returns

Double

A double-precision floating point number whose bits are identical to `value`.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Boolean Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a Boolean (`true` or `false`) value.

```
C#

public readonly struct Boolean : IComparable, IComparable<bool>,
IConvertible, IEquatable<bool>, IParsable<bool>, ISpanParsable<bool>
```

Inheritance Object → ValueType → Boolean

Implements IComparable , IComparable<Boolean> , IConvertible , IEquatable<Boolean> , IParsable<Boolean> , IParsable<TSelf> , ISpanParsable<Boolean>

Remarks

For more information about this API, see Supplemental API remarks for Boolean.

Fields

Expand table

FalseString	Represents the Boolean value `false` as a string. This field is read-only.
TrueString	Represents the Boolean value `true` as a string. This field is read-only.

Methods

Expand table

CompareTo(Boolean)	Compares this instance to a specified Boolean object and returns an integer that indicates their relationship to one another.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates their relationship to one another.
Equals(Boolean)	Returns a value indicating whether this instance is equal to a specified Boolean object.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the type code for the Boolean value type.
Parse(ReadOnlySpan<Char>)	Converts the specified span representation of a logical value to its Boolean equivalent.
Parse(String)	Converts the specified string representation of a logical value to its Boolean equivalent.
ToString()	Converts the value of this instance to its equivalent string representation (either "True" or "False").
ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation (either "True" or "False").
TryFormat(Span<Char>, Int32)	Tries to format the value of the current boolean instance into the provided span of characters.
TryParse(ReadOnly Span<Char>, Boolean)	Tries to convert the specified span representation of a logical value to its Boolean equivalent.
TryParse(String, Boolean)	Tries to convert the specified string representation of a logical value to its Boolean equivalent.

Explicit Interface Implementations

Expand table

IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).

IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IParsable<Boolean>.Parse(String, IFormatProvider)	Parses a string into a value.
IParsable<Boolean>.TryParse(String, IFormatProvider, Boolean)	
ISpanParsable<Boolean>.Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
ISpanParsable<Boolean>.TryParse(ReadOnlySpan<Char>, IFormatProvider, Boolean)	

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.FalseString Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the Boolean value `false` as a string. This field is read-only.

```
C#
public static readonly string FalseString;
```

Field Value

String

Remarks

This field is equal to the string `"False"`.

The FalseString property defines the string representation of a `false` Boolean value in formatting and parsing operations.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String

- TrueString

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.TrueString Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the Boolean value `true` as a string. This field is read-only.

C#

```
public static readonly string TrueString;
```

Field Value

String

Remarks

This field is equal to the string `"True"`.

The TrueString property defines the string representation of a `true` Boolean value in formatting and parsing operations.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String

- FalseString

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Boolean object and returns an integer that indicates their relationship to one another.

Overloads

Expand table

CompareTo(Boolean)	Compares this instance to a specified Boolean object and returns an integer that indicates their relationship to one another.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates their relationship to one another.

CompareTo(Boolean)

Compares this instance to a specified Boolean object and returns an integer that indicates their relationship to one another.

C#

```
public int CompareTo (bool value);
```

Parameters

`value` Boolean

A Boolean object to compare to this instance.

Returns

Int32

A signed integer that indicates the relative values of this instance and `value`.

Expand table

Return Value	Condition
Less than zero	This instance is `false` and `value` is `true`.
Zero	This instance and `value` are equal (either both are `true` or both are `false`).
Greater than zero	This instance is `true` and `value` is `false`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
```

```
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
```

```
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:
```

```
Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Boolean.CompareTo method because it does not have to convert the `value` parameter to an object.

If `value` has fewer bits (is narrower) than the instance type, some programming languages perform an implicit widening conversion that transforms the value of the parameter into a value with more bits.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method to compare the Int32 instance and parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- CompareTo(Object)
- Equals

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates their relationship to one another.

C#

```
public int CompareTo (object? obj);
```

Parameters

`obj` Object

An object to compare to this instance, or `null`.

Returns

Int32

A signed integer that indicates the relative order of this instance and `obj`.

Expand table

Return Value	Condition
Less than zero	This instance is `false` and `obj` is `true`.
Zero	This instance and `obj` are equal (either both are `true` or both are `false`).
Greater than zero	This instance is `true` and `obj` is `false`.

Return Value	Condition
	-or- `obj` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Boolean.

Remarks

`obj` must be `null` or an instance of `Boolean`; otherwise, an exception is thrown.

This method implements the IComparable interface.

See also

- CompareTo(Boolean)
- Equals

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Boolean.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or Boolean.

Overloads

Expand table

Equals(Boolean)	Returns a value indicating whether this instance is equal to a specified Boolean object.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(Boolean)

Returns a value indicating whether this instance is equal to a specified Boolean object.

C#

```
public bool Equals (bool obj);
```

Parameters

obj Boolean

A Boolean value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare to this instance.

Returns

Boolean

`true` if `obj` is a Boolean and has the same value as this instance; otherwise, `false`.

Remarks

This method overrides Equals.

See also

- Equals(Boolean)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A hash code for the current Boolean.

Remarks

The Boolean class implements `true` as the integer, one, and `false` as the integer, zero. However, a particular programming language might represent `true` and `false` with other values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetHashCode()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the type code for the Boolean value type.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant Boolean.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Boolean.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(ReadOnlySpan<Char>)	Converts the specified span representation of a logical value to its Boolean equivalent.
Parse(String)	Converts the specified string representation of a logical value to its Boolean equivalent.

Parse(ReadOnlySpan<Char>)

Converts the specified span representation of a logical value to its Boolean equivalent.

C#

```
public static bool Parse (ReadOnlySpan<char> value);
```

Parameters

`value` ReadOnlySpan<Char>

A span containing the characters representing the value to convert.

Returns

Boolean

`true` if `value` is equivalent to TrueString; `false` if `value` is equivalent to FalseString.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String)

Converts the specified string representation of a logical value to its Boolean equivalent.

```
C#
public static bool Parse (string value);
```

Parameters

`value` String

A string containing the value to convert.

Returns

Boolean

`true` if `value` is equivalent to TrueString; `false` if `value` is equivalent to FalseString.

Exceptions

ArgumentNullException

`value` is `null`.

FormatException

`value` is not equivalent to TrueString or FalseString.

Examples

The following code example illustrates the use of Parse method.

```
C#
```

```
bool val;
string input;

input = bool.TrueString;
val = bool.Parse(input);
Console.WriteLine("'{0}' parsed as {1}", input, val);
// The example displays the following output:
//       'True' parsed as True
```

Remarks

The `value` parameter, optionally preceded or trailed by white space, must contain either TrueString or FalseString; otherwise, an exception is thrown. The comparison is case-insensitive.

See also

- TrueString
- FalseString
- Parsing Other Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Boolean.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to its equivalent string representation (either "True" or "False").

Overloads

Expand table

ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation (either "True" or "False").
ToString()	Converts the value of this instance to its equivalent string representation (either "True" or "False").

ToString(IFormatProvider)

Converts the value of this instance to its equivalent string representation (either "True" or "False").

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

(Reserved) An IFormatProvider object.

Returns

String

TrueString if the value of this instance is `true`, or FalseString if the value of this instance is `false`.

Implements

ToString(IFormatProvider)

Remarks

The `provider` parameter is reserved. It does not participate in the execution of this method. This means that the Boolean.ToString(IFormatProvider) method, unlike most methods with a `provider` parameter, does not reflect culture-specific settings.

This method returns the constants "True" or "False". Note that XML is case-sensitive, and that the XML specification recognizes "true" and "false" as the valid set of Boolean values. If the String object returned by the ToString(IFormatProvider) method is to be written to an XML file, its String.ToLowerInvariant method should be called first to convert it to lowercase.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToString()

Converts the value of this instance to its equivalent string representation (either "True" or "False").

C#

```
public override string ToString ();
```

Returns

String

"True" (the value of the TrueString property) if the value of this instance is `true`, or "False" (the value of the FalseString property) if the value of this instance is `false`.

Examples

The following example illustrates the ToString method.

C#

```
bool raining = false;
bool busLate = true;

Console.WriteLine("raining.ToString() returns {0}", raining);
Console.WriteLine("busLate.ToString() returns {0}", busLate);
// The example displays the following output:
//       raining.ToString() returns False
//       busLate.ToString() returns True
```

Remarks

This method returns the constants "True" or "False".

Note that XML is case-sensitive, and that the XML specification recognizes "true" and "false" as the valid set of Boolean values. If the string returned by the ToString() method is to be written to an XML file, its String.ToLowerInvariant method should be called first to convert it to lowercase.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Boolean.TryFormat(Span<Char>, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current boolean instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Boolean.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, Boolean)	Tries to convert the specified span representation of a logical value to its Boolean equivalent.
TryParse(String, Boolean)	Tries to convert the specified string representation of a logical value to its Boolean equivalent.

TryParse(ReadOnlySpan<Char>, Boolean)

Tries to convert the specified span representation of a logical value to its Boolean equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<char> value, out bool result);
```

Parameters

`value` ReadOnlySpan<Char>

A span containing the characters representing the value to convert.

`result` Boolean

When this method returns, if the conversion succeeded, contains `true` if `value` is equal to TrueString or `false` if `value` is equal to FalseString. If the conversion failed, contains `false`. The conversion fails if `value` is `null` or is not equal to the value of either the TrueString or FalseString field.

Returns

Boolean

`true` if `value` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Boolean)

Tries to convert the specified string representation of a logical value to its Boolean equivalent.

C#

```
public static bool TryParse (string? value, out bool result);
```

Parameters

`value` String

A string containing the value to convert.

`result` Boolean

When this method returns, if the conversion succeeded, contains `true` if `value` is equal to TrueString or `false` if `value` is equal to FalseString. If the conversion failed, contains `false`. The conversion fails if `value` is `null` or is not equal to the value of either the TrueString or FalseString field.

Returns

Boolean

`true` if `value` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParse method to parse an array of strings. Note that the parse operation succeeds only if the string to be parsed is "True" (the value of the TrueString field) or "False" (the value of the FalseString field) in a case-insensitive comparison.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { null, String.Empty, "True", "False",
                          "true", "false", "    true    ", "0",
                          "1", "-1", "string" };
      foreach (var value in values) {
         bool flag;
         if (Boolean.TryParse(value, out flag))
            Console.WriteLine("'{0}' --> {1}", value, flag);
         else
            Console.WriteLine("Unable to parse '{0}'.",
                              value == null ? "<null>" : value);
      }
   }
}
// The example displays the following output:
//       Unable to parse '<null>'.
//       Unable to parse ''.
//       'True' --> True
//       'False' --> False
//       'true' --> True
//       'false' --> False
//       '    true    ' --> True
//       Unable to parse '0'.
//       Unable to parse '1'.
//       Unable to parse '-1'.
//       Unable to parse 'string'.
```

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails.

The `value` parameter can be preceded or followed by white space. The comparison is ordinal and case-insensitive.

See also

- Parse(String)
- TrueString
- FalseString
- Parsing Other Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` or `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Byte

1 if the value of this instance is `true`; otherwise, 0.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToByte(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

This conversion is not supported. No value is returned.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

You attempt to convert a Boolean value to a Char value. This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

You attempt to convert a Boolean value to a DateTime value. This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

```
C#
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

1 if this instance is `true`; otherwise, 0.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Double

1 if this instance is `true`; otherwise, 0.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

```
C#
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

1 if this instance is `true`; otherwise, 0.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16 method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

1 if this instance is `true`; otherwise, 0.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32 method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Int64

1 if this instance is `true`; otherwise, 0.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

SByte

1 if this instance is `true`; otherwise, 0.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative

is to call the Convert.ToSByte(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Single

1 if this instance is `true`; otherwise, 0.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The desired type.

`provider` IFormatProvider

An IFormatProvider implementation that supplies culture-specific information about the format of the returned value.

Returns

Object

An object of the specified type, with a value that is equivalent to the value of this `Boolean` object.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

The requested type conversion is not supported.

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Convert

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

UInt16

1 if this instance is `true`; otherwise, 0.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative

is to call the Convert.ToUInt16(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt16
- ToUInt16(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

UInt32

1 if this instance is `true`; otherwise, 0.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative

is to call the Convert.ToUInt32(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt32
- ToUInt32(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

1 if this instance is `true`; otherwise, 0.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Boolean instance is cast to an IConvertible interface. The recommended alternative

is to call the Convert.ToUInt64(Boolean) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt64
- ToUInt64(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Boolean.IParsable<Boolean>.Parse(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a string into a value.

```
C#
static bool IParsable<bool>.Parse (string s, IFormatProvider provider);
```

Parameters

s String

The string to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Boolean

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Boolean.IParsable<Boolean>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IParsable<bool>.TryParse (string s, IFormatProvider provider,
out bool result);
```

Parameters

`s` String

`provider` IFormatProvider

`result` Boolean

Returns

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Boolean.ISpanParsable<Boolean>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a span of characters into a value.

```
C#

static bool ISpanParsable<bool>.Parse (ReadOnlySpan<char> s, IFormatProvider
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Boolean

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Boolean.ISpanParsable<Boolean>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool ISpanParsable<bool>.TryParse (ReadOnlySpan<char> s,
IFormatProvider provider, out bool result);
```

Parameters

`s` ReadOnlySpan<Char>

`provider` IFormatProvider

`result` Boolean

Returns

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Buffer Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Manipulates arrays of primitive types.

C#

```
public static class Buffer
```

Inheritance Object → Buffer

Examples

The following code example illustrates the use of several `Buffer` class methods.

C#

```
// Example of the Buffer class methods.
using System;

class BufferClassDemo
{
    // Display the array elements from right to left in hexadecimal.
    public static void DisplayArray( short[ ] arr )
    {
        Console.Write( "  arr:" );
        for( int loopX = arr.Length - 1; loopX >= 0; loopX-- )
            Console.Write( " {0:X4}", arr[ loopX ] );
        Console.WriteLine( );
    }

    public static void Main( )
    {
        // This array is to be modified and displayed.
        short[ ] arr = { 258, 259, 260, 261, 262, 263, 264,
                         265, 266, 267, 268, 269, 270, 271 };

        Console.WriteLine( "This example of the Buffer class " +
            "methods generates the following output.\n" +
            "Note: The array is displayed from right to left.\n" );
```

```
        Console.WriteLine( "Initial values of array:\n" );

        // Display the initial array values and ByteLength.
        DisplayArray( arr );
        Console.WriteLine( "\nBuffer.ByteLength( arr ): {0}",
            Buffer.ByteLength( arr ) );

        // Copy a region of the array; set a byte within the array.
        Console.WriteLine( "\nCall these methods: \n" +
            "  Buffer.BlockCopy( arr, 5, arr, 16, 9 ),\n" +
            "  Buffer.SetByte( arr, 7, 170 ).\n" );

        Buffer.BlockCopy( arr, 5, arr, 16, 9 );
        Buffer.SetByte( arr, 7, 170 );

        // Display the array and a byte within the array.
        Console.WriteLine( "Final values of array:\n" );
        DisplayArray( arr );
        Console.WriteLine( "\nBuffer.GetByte( arr, 26 ): {0}",
            Buffer.GetByte( arr, 26 ) );
    }
}

/*
This example of the Buffer class methods generates the following output.
Note: The array is displayed from right to left.

Initial values of array:

  arr: 010F 010E 010D 010C 010B 010A 0109 0108 0107 0106 0105 0104 0103 0102

Buffer.ByteLength( arr ): 28

Call these methods:
  Buffer.BlockCopy( arr, 5, arr, 16, 9 ),
  Buffer.SetByte( arr, 7, 170 ).

Final values of array:

  arr: 010F 0101 0801 0701 0601 0501 0109 0108 0107 0106 AA05 0104 0103 0102

Buffer.GetByte( arr, 26 ): 15
*/
```

Remarks

Buffer only affects arrays of primitive types; this class does not apply to objects. Each primitive type is treated as a series of bytes without regard to any behavior or limitation associated with the primitive type.

Buffer provides methods to copy bytes from one array of primitive types to another array of primitive types, get a byte from an array, set a byte in an array, and obtain the length of an array. This class provides better performance for manipulating primitive types than similar methods in the System.Array class.

Buffer is applicable to the following primitive types: Boolean, Char, SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64, IntPtr, UIntPtr, Single, and Double.

Methods

Expand table

BlockCopy(Array, Int32, Array, Int32, Int32)	Copies a specified number of bytes from a source array starting at a particular offset to a destination array starting at a particular offset.
ByteLength(Array)	Returns the number of bytes in the specified array.
GetByte(Array, Int32)	Retrieves the byte at the specified location in the specified array.
MemoryCopy(Void*, Void*, Int64, Int64)	Copies a number of bytes specified as a long integer value from one address in memory to another. This API is not CLS-compliant.
MemoryCopy(Void*, Void*, UInt64, UInt64)	Copies a number of bytes specified as an unsigned long integer value from one address in memory to another. This API is not CLS-compliant.
SetByte(Array, Int32, Byte)	Assigns a specified value to a byte at a particular location in a specified array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Buffer.BlockCopy(Array, Int32, Array, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies a specified number of bytes from a source array starting at a particular offset to a destination array starting at a particular offset.

C#

```
public static void BlockCopy (Array src, int srcOffset, Array dst, int
dstOffset, int count);
```

Parameters

src Array

The source buffer.

srcOffset Int32

The zero-based byte offset into `src`.

dst Array

The destination buffer.

dstOffset Int32

The zero-based byte offset into `dst`.

count Int32

The number of bytes to copy.

Exceptions

ArgumentNullException

`src` or `dst` is `null`.

ArgumentException

`src` or `dst` is not an array of primitives.

-or-

The number of bytes in `src` is less than `srcOffset` plus `count`.

-or-

The number of bytes in `dst` is less than `dstOffset` plus `count`.

ArgumentOutOfRangeException

`srcOffset`, `dstOffset`, or `count` is less than 0.

Examples

The following example copies regions of arrays by using the BlockCopy method. For each BlockCopy operation, it lists the source and destination arrays as both an array of values and as a sequence of bytes. The example illustrates the importance of considering a system's endianness when working with the BlockCopy method: Because Windows systems are little-endian, the lower-order bytes of a primitive data type's value precede the higher-order bytes.

C#

```
using System;

class Example
{
    // Display the individual bytes in the array in hexadecimal.
    public static void DisplayArray(Array arr, string name)
    {
        Console.WindowWidth = 120;
        Console.Write("{0,11}:", name);
        for (int ctr = 0; ctr < arr.Length; ctr++)
        {
            byte[] bytes;
            if (arr is long[])
               bytes = BitConverter.GetBytes((long) arr.GetValue(ctr));
            else
               bytes = BitConverter.GetBytes((short) arr.GetValue(ctr));

            foreach (byte byteValue in bytes)
               Console.Write(" {0:X2}", byteValue);
        }
        Console.WriteLine();
    }
```

```
    // Display the individual array element values in hexadecimal.
    public static void DisplayArrayValues(Array arr, string name)
    {
        // Get the length of one element in the array.
        int elementLength = Buffer.ByteLength(arr) / arr.Length;
        string formatString = String.Format(" {{0:X{0}}}", 2 *
elementLength);
        Console.Write( "{0,11}:", name);
        for (int ctr = 0; ctr < arr.Length; ctr++)
            Console.Write(formatString, arr.GetValue(ctr));

        Console.WriteLine();
    }

    public static void Main( )
    {
        // These are the source and destination arrays for BlockCopy.
        short[] src  = { 258, 259, 260, 261, 262, 263, 264,
                         265, 266, 267, 268, 269, 270 };
        long[] dest = { 17, 18, 19, 20 };

        // Display the initial value of the arrays in memory.
        Console.WriteLine( "Initial values of arrays:");
        Console.WriteLine("   Array values as Bytes:");
        DisplayArray(src, "src" );
        DisplayArray(dest, "dest");
        Console.WriteLine("   Array values:");
        DisplayArrayValues(src, "src");
        DisplayArrayValues(dest, "dest");
        Console.WriteLine();

        // Copy bytes 5-10 from source to index 7 in destination and display
the result.
        Buffer.BlockCopy(src, 5, dest, 7, 6);
        Console.WriteLine("Buffer.BlockCopy(src, 5, dest, 7, 6 )");
        Console.WriteLine("   Array values as Bytes:");
        DisplayArray(src, "src");
        DisplayArray(dest, "dest");
        Console.WriteLine("   Array values:");
        DisplayArrayValues(src, "src");
        DisplayArrayValues(dest, "dest");
        Console.WriteLine();

        // Copy bytes 16-20 from source to index 22 in destination and
display the result.
        Buffer.BlockCopy(src, 16, dest, 22, 5);
        Console.WriteLine("Buffer.BlockCopy(src, 16, dest, 22, 5)");
        Console.WriteLine("   Array values as Bytes:");
        DisplayArray(src, "src");
        DisplayArray(dest, "dest");
        Console.WriteLine("   Array values:");
        DisplayArrayValues(src, "src");
        DisplayArrayValues(dest, "dest");
        Console.WriteLine();
```

```
        // Copy overlapping range of bytes 4-10 to index 5 in source.
        Buffer.BlockCopy(src, 4, src, 5, 7 );
        Console.WriteLine("Buffer.BlockCopy( src, 4, src, 5, 7)");
        Console.WriteLine("   Array values as Bytes:");
        DisplayArray(src, "src");
        DisplayArray(dest, "dest");
        Console.WriteLine("   Array values:");
        DisplayArrayValues(src, "src");
        DisplayArrayValues(dest, "dest");
        Console.WriteLine();

        // Copy overlapping range of bytes 16-22 to index 15 in source.
        Buffer.BlockCopy(src, 16, src, 15, 7);
        Console.WriteLine("Buffer.BlockCopy( src, 16, src, 15, 7)");
        Console.WriteLine("   Array values as Bytes:");
        DisplayArray(src, "src");
        DisplayArray(dest, "dest");
        Console.WriteLine("   Array values:");
        DisplayArrayValues(src, "src");
        DisplayArrayValues(dest, "dest");
    }
}
// The example displays the following output:
//    Initial values of arrays:
//       Array values as Bytes:
//            src: 02 01 03 01 04 01 05 01 06 01 07 01 08 01 09 01 0A 01 0B
01 0C 01 0D 01 0E 01
//           dest: 11 00 00 00 00 00 00 00 12 00 00 00 00 00 00 00 13 00 00
00 00 00 00 00 14 00 00 00 00 00 00 00
//       Array values:
//            src: 0102 0103 0104 0105 0106 0107 0108 0109 010A 010B 010C
010D 010E
//           dest: 0000000000000011 0000000000000012 0000000000000013
0000000000000014
//
//    Buffer.BlockCopy(src, 5, dest, 7, 6 )
//       Array values as Bytes:
//            src: 02 01 03 01 04 01 05 01 06 01 07 01 08 01 09 01 0A 01 0B
01 0C 01 0D 01 0E 01
//           dest: 11 00 00 00 00 00 00 01 05 01 06 01 07 00 00 00 13 00 00
00 00 00 00 00 14 00 00 00 00 00 00 00
//       Array values:
//            src: 0102 0103 0104 0105 0106 0107 0108 0109 010A 010B 010C
010D 010E
//           dest: 0100000000000011 0000000701060105 0000000000000013
0000000000000014
//
//    Buffer.BlockCopy(src, 16, dest, 22, 5)
//       Array values as Bytes:
//            src: 02 01 03 01 04 01 05 01 06 01 07 01 08 01 09 01 0A 01 0B
01 0C 01 0D 01 0E 01
//           dest: 11 00 00 00 00 00 00 01 05 01 06 01 07 00 00 00 13 00 00
00 00 00 0A 01 0B 01 0C 00 00 00 00 00
//       Array values:
//            src: 0102 0103 0104 0105 0106 0107 0108 0109 010A 010B 010C
```

```
010D 010E
//           dest: 0100000000000011 0000000701060105 010A000000000013
00000000000C010B
//
//    Buffer.BlockCopy( src, 4, src, 5, 7)
//       Array values as Bytes:
//            src: 02 01 03 01 04 04 01 05 01 06 01 07 08 01 09 01 0A 01 0B
01 0C 01 0D 01 0E 01
//           dest: 11 00 00 00 00 00 00 01 05 01 06 01 07 00 00 00 13 00 00
00 00 00 0A 01 0B 01 0C 00 00 00 00 00
//       Array values:
//            src: 0102 0103 0404 0501 0601 0701 0108 0109 010A 010B 010C
010D 010E
//           dest: 0100000000000011 0000000701060105 010A000000000013
00000000000C010B
//
//    Buffer.BlockCopy( src, 16, src, 15, 7)
//       Array values as Bytes:
//            src: 02 01 03 01 04 04 01 05 01 06 01 07 08 01 09 0A 01 0B 01
0C 01 0D 0D 01 0E 01
//           dest: 11 00 00 00 00 00 00 01 05 01 06 01 07 00 00 00 13 00 00
00 00 00 0A 01 0B 01 0C 00 00 00 00 00
//       Array values:
//            src: 0102 0103 0404 0501 0601 0701 0108 0A09 0B01 0C01 0D01
010D 010E
//           dest: 0100000000000011 0000000701060105 010A000000000013
00000000000C010B
```

Remarks

This method copies `count` bytes from `src`, beginning at `srcOffset`, to `dst`, beginning at `dstOffset`. Both `srcOffset` and `dstOffset` are zero-based; that is, the first byte in each buffer is at position 0, not position 1.

The BlockCopy method accesses the bytes in the `src` parameter array using offsets into memory, not programming constructs such as indexes or upper and lower array bounds. For example, if in the programming language of your application you declare an Int32 array with a zero-based lower bound of -50, and then pass the array and an offset of 5 to the BlockCopy method, the first array element the method will access is the second element of the array, which is at index -49. Furthermore, which byte of array element -49 is accessed first depends on the endianness of the computer that is executing your application.

As its name suggests, the BlockCopy method copies a block of bytes as a whole, rather than copying one byte at a time. Therefore, if `src` and `dst` reference the same array, and the range from `srcOffset` + `count` -1 overlaps the range from `dstOffset` + `count` - 1, the values of the overlapping bytes are not overwritten before they are copied to the

destination. In the following example, the values of bytes 0-16 in an array named `arr` are copied to bytes 12-28. Despite the overlapping range, the values of the source bytes are successfully copied.

C#

```
const int INT_SIZE = 4;
int[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
Buffer.BlockCopy(arr, 0 * INT_SIZE, arr, 3 * INT_SIZE, 4 * INT_SIZE);
foreach (int value in arr)
   Console.Write("{0}  ", value);
// The example displays the following output:
//       2  4  6  2  4  6  8  16  18  20
```

In the following example, the values of bytes 12-28 in an array named `arr` are copied to bytes 0-16. Again, despite the overlapping range, the values of the source bytes are successfully copied.

C#

```
const int INT_SIZE = 4;
int[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
Buffer.BlockCopy(arr, 3 * INT_SIZE, arr, 0 * INT_SIZE, 4 * INT_SIZE);
foreach (int value in arr)
   Console.Write("{0}  ", value);
// The example displays the following output:
//       8  10  12  14  10  12  14  16  18  20
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Buffer.ByteLength(Array) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of bytes in the specified array.

```
C#
public static int ByteLength (Array array);
```

Parameters

array Array

An array.

Returns

Int32

The number of bytes in the array.

Exceptions

ArgumentNullException

`array` is `null`.

ArgumentException

`array` is not a primitive.

OverflowException

`array` is larger than 2 gigabytes (GB).

Examples

The following code example illustrates the use of the `ByteLength` method to return the number of bytes in an array.

C#

```
// Example of the Buffer.ByteLength method.
using System;

class ByteLengthDemo
{
    const string formatter = "{0,10}{1,20}{2,9}{3,12}";

    public static void ArrayInfo( Array arr, string name )
    {
        int byteLength = Buffer.ByteLength( arr );

        // Display the array name, type, Length, and ByteLength.
        Console.WriteLine( formatter, name, arr.GetType( ),
            arr.Length, byteLength );
    }

    public static void Main( )
    {
        byte[ ]   bytes   = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
        bool[ ]   bools   = { true, false, true, false, true };
        char[ ]   chars   = { ' ', '$', '\"', 'A', '{' };
        short[ ]  shorts  = { 258, 259, 260, 261, 262, 263 };
        float[ ]  singles = { 1, 678, 2.37E33F, .00415F, 8.9F };
        double[ ] doubles = { 2E-22, .003, 4.4E44, 555E55 };
        long[ ]   longs   = { 1, 10, 100, 1000, 10000, 100000 };

        Console.WriteLine(
            "This example of the Buffer.ByteLength( Array ) " +
            "\nmethod generates the following output.\n" );
        Console.WriteLine( formatter, "Array name", "Array type",
            "Length", "ByteLength" );
        Console.WriteLine( formatter, "----------", "----------",
            "------", "----------" );

        // Display the Length and ByteLength for each array.
        ArrayInfo( bytes, "bytes" );
        ArrayInfo( bools, "bools" );
        ArrayInfo( chars, "chars" );
        ArrayInfo( shorts, "shorts" );
        ArrayInfo( singles, "singles" );
        ArrayInfo( doubles, "doubles" );
        ArrayInfo( longs, "longs" );
    }
}

/*
This example of the Buffer.ByteLength( Array )
method generates the following output.
```

```
Array name          Array type   Length  ByteLength
----------          ----------   ------  ----------
     bytes       System.Byte[]       10          10
     bools    System.Boolean[]        5           5
     chars       System.Char[]        5          10
    shorts      System.Int16[]        6          12
   singles     System.Single[]        5          20
   doubles     System.Double[]        4          32
     longs      System.Int64[]        6          48
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Buffer.GetByte(Array, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the byte at the specified location in the specified array.

C#

```
public static byte GetByte (Array array, int index);
```

Parameters

array Array

An array.

index Int32

A location in the array.

Returns

Byte

The byte at the specified location in the array.

Exceptions

ArgumentException

`array` is not a primitive.

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is negative or greater than the length of `array`.

OverflowException

`array` is larger than 2 gigabytes (GB).

Examples

The following code example displays the byte values at specified locations within arrays using the `GetByte` method.

C#

```
// Example of the Buffer.GetByte method.
using System;

class GetByteDemo
{
    const string formatter = "{0,10}{1,10}{2,9} {3}";

    // Display the array contents in hexadecimal.
    public static void DisplayArray( Array arr, string name )
    {
        // Get the array element width; format the formatting string.
        int elemWidth = Buffer.ByteLength( arr ) / arr.Length;
        string format = String.Format( " {{0:X{0}}}", 2 * elemWidth );

        // Display the array elements from right to left.
        Console.Write( "{0,5}:", name );
        for( int loopX = arr.Length - 1; loopX >= 0; loopX-- )
            Console.Write( format, arr.GetValue( loopX ) );
        Console.WriteLine( );
    }

    public static void ArrayInfo( Array arr, string name, int index )
    {
        byte value = Buffer.GetByte( arr, index );

        // Display the array name, index, and byte to be viewed.
        Console.WriteLine( formatter, name, index, value,
            String.Format( "0x{0:X2}", value ) );
    }

    public static void Main( )
    {
        // These are the arrays to be viewed with GetByte.
        long[ ] longs =
            { 333333333333333333, 666666666666666666, 999999999999999999 };
        int[ ]  ints  =
            { 111111111, 222222222, 333333333, 444444444, 555555555 };

        Console.WriteLine( "This example of the " +
            "Buffer.GetByte( Array, int ) \n" +
            "method generates the following output.\n" +
            "Note: The arrays are displayed from right to left.\n" );
        Console.WriteLine( "  Values of arrays:\n" );
```

```
        // Display the values of the arrays.
        DisplayArray( longs, "longs" );
        DisplayArray( ints, "ints" );
        Console.WriteLine( );

        Console.WriteLine( formatter, "Array", "index", "value", "" );
        Console.WriteLine( formatter, "-----", "-----", "-----",
            "----" );

        // Display the Length and ByteLength for each array.
        ArrayInfo( ints, "ints", 0 );
        ArrayInfo( ints, "ints", 7 );
        ArrayInfo( ints, "ints", 10 );
        ArrayInfo( ints, "ints", 17 );
        ArrayInfo( longs, "longs", 0 );
        ArrayInfo( longs, "longs", 6 );
        ArrayInfo( longs, "longs", 10 );
        ArrayInfo( longs, "longs", 17 );
        ArrayInfo( longs, "longs", 21 );
    }
}

/*
This example of the Buffer.GetByte( Array, int )
method generates the following output.
Note: The arrays are displayed from right to left.

  Values of arrays:

longs: 0DE0B6B3A763FFFF 094079CD1A42AAAA 04A03CE68D215555
 ints: 211D1AE3 1A7DAF1C 13DE4355 0D3ED78E 069F6BC7

     Array     index    value
     -----     -----    ----- ----
      ints         0      199 0xC7
      ints         7       13 0x0D
      ints        10      222 0xDE
      ints        17       26 0x1A
     longs         0       85 0x55
     longs         6      160 0xA0
     longs        10       66 0x42
     longs        17      255 0xFF
     longs        21      182 0xB6
*/
```

Remarks

The `GetByte` method gets a particular byte out of the array. The array must be an array of primitives.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Buffer.MemoryCopy Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies a block of memory.

Overloads

Expand table

MemoryCopy(Void*, Void*, Int64, Int64)	Copies a number of bytes specified as a long integer value from one address in memory to another. This API is not CLS-compliant.
MemoryCopy(Void*, Void*, UInt64, UInt64)	Copies a number of bytes specified as an unsigned long integer value from one address in memory to another. This API is not CLS-compliant.

MemoryCopy(Void*, Void*, Int64, Int64)

> Important
>
> This API is not CLS-compliant.

Copies a number of bytes specified as a long integer value from one address in memory to another.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static void MemoryCopy (void* source, void* destination, long
destinationSizeInBytes, long sourceBytesToCopy);
```

Parameters

`source` Void*

The address of the bytes to copy.

`destination` Void*

The target address.

`destinationSizeInBytes` Int64

The number of bytes available in the destination memory block.

`sourceBytesToCopy` Int64

The number of bytes to copy.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

`sourceBytesToCopy` is greater than `destinationSizeInBytes`.

Remarks

This method copies `sourceBytesToCopy` bytes from the address specified by `source` to the address specified by `destination`. If some regions of the source area and the destination overlap, the function ensures that the original source bytes in the overlapping region are copied before being overwritten.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MemoryCopy(Void*, Void*, UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Copies a number of bytes specified as an unsigned long integer value from one address in memory to another.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static void MemoryCopy (void* source, void* destination, ulong
destinationSizeInBytes, ulong sourceBytesToCopy);
```

Parameters

`source` Void*

The address of the bytes to copy.

`destination` Void*

The target address.

`destinationSizeInBytes` UInt64

The number of bytes available in the destination memory block.

`sourceBytesToCopy` UInt64

The number of bytes to copy.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

`sourceBytesToCopy` is greater than `destinationSizeInBytes`.

Remarks

This method copies `sourceBytesToCopy` bytes from the address specified by `source` to the address specified by `destination`. If some regions of the source area and the destination overlap, the function ensures that the original source bytes in the overlapping region are copied before being overwritten.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Buffer.SetByte(Array, Int32, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Assigns a specified value to a byte at a particular location in a specified array.

C#

```
public static void SetByte (Array array, int index, byte value);
```

Parameters

array Array

An array.

index Int32

A location in the array.

value Byte

A value to assign.

Exceptions

ArgumentException

`array` is not a primitive.

ArgumentNullException

`array` is `null`.

ArgumentOutOfRangeException

`index` is negative or greater than the length of `array`.

OverflowException

`array` is larger than 2 gigabytes (GB).

Examples

The following code example assigns values to bytes at particular locations within arrays using the `SetByte` method.

C#

```
// Example of the Buffer.SetByte method.
using System;

class SetByteDemo
{
    // Display the array contents in hexadecimal.
    public static void DisplayArray( Array arr, string name )
    {
        // Get the array element width; format the formatting string.
        int elemWidth = Buffer.ByteLength( arr ) / arr.Length;
        string format = String.Format( " {{0:X{0}}}", 2 * elemWidth );

        // Display the array elements from right to left.
        Console.Write( "{0,7}:", name );
        for( int loopX = arr.Length - 1; loopX >= 0; loopX-- )
            Console.Write( format, arr.GetValue( loopX ) );
        Console.WriteLine( );
    }

    public static void Main( )
    {
        // These are the arrays to be modified with SetByte.
        short[ ] shorts = new short[ 10 ];
        long[ ]  longs  = new long[ 3 ];

        Console.WriteLine( "This example of the " +
            "Buffer.SetByte( Array, int, byte ) \n" +
            "method generates the following output.\n" +
            "Note: The arrays are displayed from right to left.\n" );
        Console.WriteLine( "  Initial values of arrays:\n" );

        // Display the initial values of the arrays.
        DisplayArray( shorts, "shorts" );
        DisplayArray( longs, "longs" );

        // Copy two regions of source array to destination array,
        // and two overlapped copies from source to source.
        Console.WriteLine( "\n" +
            "  Array values after setting byte 3 = 25, \n" +
            "  byte 6 = 64, byte 12 = 121, and byte 17 = 196:\n" );

        Buffer.SetByte( shorts, 3, 25 );
        Buffer.SetByte( shorts, 6, 64 );
```

```
        Buffer.SetByte( shorts, 12, 121 );
        Buffer.SetByte( shorts, 17, 196 );
        Buffer.SetByte( longs, 3, 25 );
        Buffer.SetByte( longs, 6, 64 );
        Buffer.SetByte( longs, 12, 121 );
        Buffer.SetByte( longs, 17, 196 );

        // Display the arrays again.
        DisplayArray( shorts, "shorts" );
        DisplayArray( longs, "longs" );
    }
}

/*
This example of the Buffer.SetByte( Array, int, byte )
method generates the following output.
Note: The arrays are displayed from right to left.

  Initial values of arrays:

 shorts: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  longs: 0000000000000000 0000000000000000 0000000000000000

  Array values after setting byte 3 = 25,
  byte 6 = 64, byte 12 = 121, and byte 17 = 196:

 shorts: 0000 C400 0000 0079 0000 0000 0040 0000 1900 0000
  longs: 000000000000C400 0000007900000000 0040000019000000
*/
```

Remarks

`array` must be an array of primitives.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an 8-bit unsigned integer.

C#

```
public readonly struct Byte : IComparable<byte>, IConvertible,
IEquatable<byte>, IParsable<byte>, ISpanParsable<byte>,
IUtf8SpanParsable<byte>, System.Numerics.IAdditionOperators<byte,byte,byte>,
System.Numerics.IAdditiveIdentity<byte,byte>,
System.Numerics.IBinaryInteger<byte>, System.Numerics.IBinaryNumber<byte>,
System.Numerics.IBitwiseOperators<byte,byte,byte>,
System.Numerics.IComparisonOperators<byte,byte,bool>,
System.Numerics.IDecrementOperators<byte>,
System.Numerics.IDivisionOperators<byte,byte,byte>,
System.Numerics.IEqualityOperators<byte,byte,bool>,
System.Numerics.IIncrementOperators<byte>,
System.Numerics.IMinMaxValue<byte>,
System.Numerics.IModulusOperators<byte,byte,byte>,
System.Numerics.IMultiplicativeIdentity<byte,byte>,
System.Numerics.IMultiplyOperators<byte,byte,byte>,
System.Numerics.INumber<byte>, System.Numerics.INumberBase<byte>,
System.Numerics.IShiftOperators<byte,int,byte>,
System.Numerics.ISubtractionOperators<byte,byte,byte>,
System.Numerics.IUnaryNegationOperators<byte,byte>,
System.Numerics.IUnaryPlusOperators<byte,byte>,
System.Numerics.IUnsignedNumber<byte>
```

Inheritance Object → ValueType → Byte

Implements IComparable , IComparable<Byte> , IConvertible , IEquatable<Byte> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Byte> , IParsable<TSelf> , ISpanParsable<Byte> , ISpanParsable<TSelf> , IAdditionOperators<Byte,Byte,Byte> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Byte,Byte> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Byte> ,

IBinaryNumber<Byte> , IBinaryNumber<TSelf> ,
IBitwiseOperators<Byte,Byte,Byte> , IBitwiseOperators<TSelf,TSelf,TSelf> ,
IComparisonOperators<Byte,Byte,Boolean> ,
IComparisonOperators<TSelf,TSelf,Boolean> ,
IDecrementOperators<Byte> , IDecrementOperators<TSelf> ,
IDivisionOperators<Byte,Byte,Byte> , IDivisionOperators<TSelf,TSelf,TSelf> ,
IEqualityOperators<Byte,Byte,Boolean> ,
IEqualityOperators<TSelf,TOther,TResult> ,
IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Byte> ,
IIncrementOperators<TSelf> , IMinMaxValue<Byte> ,
IModulusOperators<Byte,Byte,Byte> ,
IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<Byte,Byte> ,
IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<Byte,Byte,Byte> ,
IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Byte> , INumber<TSelf> ,
INumberBase<Byte> , INumberBase<TSelf> ,
IShiftOperators<Byte,Int32,Byte> , IShiftOperators<TSelf,Int32,TSelf> ,
ISubtractionOperators<Byte,Byte,Byte> ,
ISubtractionOperators<TSelf,TSelf,TSelf> ,
IUnaryNegationOperators<Byte,Byte> ,
IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<Byte,Byte> ,
IUnaryPlusOperators<TSelf,TSelf> , IUnsignedNumber<Byte> ,
IUtf8SpanFormattable , IUtf8SpanParsable<Byte> ,
IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Byte.

Fields

Expand table

MaxValue	Represents the largest possible value of a Byte. This field is constant.

MinValue	Represents the smallest possible value of a Byte. This field is constant.

Methods

Expand table

Clamp(Byte, Byte, Byte)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Byte)	Compares this instance to a specified 8-bit unsigned integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(Byte, Byte)	Computes the quotient and remainder of two values.
Equals(Byte)	Returns a value indicating whether this instance and a specified Byte object represent the same value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Byte.
IsEvenInteger(Byte)	Determines if a value represents an even integral number.
IsOddInteger(Byte)	Determines if a value represents an odd integral number.
IsPow2(Byte)	Determines if a value is a power of two.
LeadingZeroCount(Byte)	Computes the number of leading zeros in a value.
Log2(Byte)	Computes the log2 of a value.

Max(Byte, Byte)	Compares two values to compute which is greater.
Min(Byte, Byte)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent.
Parse(String)	Converts the string representation of a number to its Byte equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its Byte equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its Byte equivalent.
Parse(String, NumberStyles, IFormat Provider)	Converts the string representation of a number in a specified style and culture-specific format to its Byte equivalent.
PopCount(Byte)	Computes the number of bits that are set in a value.
RotateLeft(Byte, Int32)	Rotates a value left by a given amount.
RotateRight(Byte, Int32)	Rotates a value right by a given amount.
Sign(Byte)	Computes the sign of a value.
ToString()	Converts the value of the current Byte object to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of the current Byte object to its equivalent string representation using the specified culture-specific formatting information.
ToString(String)	Converts the value of the current Byte object to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the value of the current Byte object to its equivalent string representation using the specified format and culture-specific formatting information.

TrailingZeroCount(Byte)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current 8-bit unsigned integer instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, Byte)	Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Byte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Byte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Byte)	Tries to convert the span representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Byte)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Byte)	Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Byte)	Tries to convert the string representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, Byte)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Byte)	Converts the string representation of a number in a specified style and culture-specific format to its Byte equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<Byte,Byte,Byte>.Addition(Byte, Byte)	Adds two values together to compute their sum.

IAdditionOperators<Byte,Byte,Byte>.CheckedAddition(Byte, Byte)	Adds two values together to compute their sum.
IAdditiveIdentity<Byte,Byte>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<Byte>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Byte>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Byte>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Byte)	
IBinaryInteger<Byte>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Byte)	
IBinaryInteger<Byte>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Byte>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Byte>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Byte,Byte,Byte>.BitwiseAnd(Byte, Byte)	Computes the bitwise-and of two values.
IBitwiseOperators<Byte,Byte,Byte>.BitwiseOr(Byte, Byte)	Computes the bitwise-or of two values.
IBitwiseOperators<Byte,Byte,Byte>.ExclusiveOr(Byte, Byte)	Computes the exclusive-or of two values.
IBitwiseOperators<Byte,Byte,Byte>.OnesComplement(Byte)	Computes the ones-complement representation of a given value.
IComparisonOperators<Byte,Byte,Boolean>.GreaterThan(Byte, Byte)	Compares two values to determine which is greater.
IComparisonOperators<Byte,Byte,Boolean>.GreaterThanOrEqual(Byte, Byte)	Compares two values to determine which is greater or equal.
IComparisonOperators<Byte,Byte,Boolean>.LessThan(Byte, Byte)	Compares two values to determine which is less.
IComparisonOperators<Byte,Byte,Boolean>.LessThanOrEqual(Byte, Byte)	Compares two values to determine which is less or equal.

IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Byte>.Checked Decrement(Byte)	Decrements a value.
IDecrementOperators<Byte>.Decrement(Byte)	Decrements a value.

IDivisionOperators<Byte,Byte,Byte>.Division(Byte, Byte)	Divides one value by another to compute their quotient.
IEqualityOperators<Byte,Byte,Boolean>.Equality(Byte, Byte)	Compares two values to determine equality.
IEquality Operators<Byte,Byte,Boolean>.Inequality(Byte, Byte)	Compares two values to determine inequality.
IIncrementOperators<Byte>.CheckedIncrement(Byte)	Increments a value.
IIncrementOperators<Byte>.Increment(Byte)	Increments a value.
IMinMaxValue<Byte>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Byte>.MinValue	Gets the minimum value of the current type.
IModulusOperators<Byte,Byte,Byte>.Modulus(Byte, Byte)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Byte,Byte>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<Byte,Byte,Byte>.Checked Multiply(Byte, Byte)	Multiplies two values together to compute their product.
IMultiplyOperators<Byte,Byte,Byte>.Multiply(Byte, Byte)	Multiplies two values together to compute their product.
INumber<Byte>.CopySign(Byte, Byte)	Copies the sign of a value to the sign of another value.
INumber<Byte>.MaxNumber(Byte, Byte)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<Byte>.MinNumber(Byte, Byte)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Byte>.Abs(Byte)	Computes the absolute of a value.
INumberBase<Byte>.IsCanonical(Byte)	Determines if a value is in its canonical representation.
INumberBase<Byte>.IsComplexNumber(Byte)	Determines if a value represents a complex number.
INumberBase<Byte>.IsFinite(Byte)	Determines if a value is finite.

INumberBase<Byte>.IsImaginaryNumber(Byte)	Determines if a value represents a pure imaginary number.
INumberBase<Byte>.IsInfinity(Byte)	Determines if a value is infinite.
INumberBase<Byte>.IsInteger(Byte)	Determines if a value represents an integral number.
INumberBase<Byte>.IsNaN(Byte)	Determines if a value is NaN.
INumberBase<Byte>.IsNegative(Byte)	Determines if a value is negative.
INumberBase<Byte>.IsNegativeInfinity(Byte)	Determines if a value is negative infinity.
INumberBase<Byte>.IsNormal(Byte)	Determines if a value is normal.
INumberBase<Byte>.IsPositive(Byte)	Determines if a value is positive.
INumberBase<Byte>.IsPositiveInfinity(Byte)	Determines if a value is positive infinity.
INumberBase<Byte>.IsRealNumber(Byte)	Determines if a value represents a real number.
INumberBase<Byte>.IsSubnormal(Byte)	Determines if a value is subnormal.
INumberBase<Byte>.IsZero(Byte)	Determines if a value is zero.
INumberBase<Byte>.MaxMagnitude(Byte, Byte)	Compares two values to compute which is greater.
INumberBase<Byte>.MaxMagnitudeNumber(Byte, Byte)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Byte>.MinMagnitude(Byte, Byte)	Compares two values to compute which is lesser.
INumberBase<Byte>.MinMagnitudeNumber(Byte, Byte)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Byte>.One	Gets the value `1` for the type.
INumberBase<Byte>.Radix	Gets the radix, or base, for the type.
INumberBase<Byte>.TryConvertFromChecked<TOther>(TOther, Byte)	
INumberBase<Byte>.TryConvertFromSaturating<TOther>(TOther, Byte)	
INumberBase<Byte>.TryConvertFromTruncating<TOther>(TOther, Byte)	
INumberBase<Byte>.TryConvertToChecked<TOther>(Byte, TOther)	Tries to convert an instance of the the current type to another type, throwing an

	overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Byte>.TryConvertToSaturating<TOther>(Byte, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Byte>.TryConvertToTruncating<TOther>(Byte, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Byte>.Zero	Gets the value `0` for the type.
IShiftOperators<Byte,Int32,Byte>.LeftShift(Byte, Int32)	Shifts a value left by a given amount.
IShiftOperators<Byte,Int32,Byte>.RightShift(Byte, Int32)	Shifts a value right by a given amount.
IShiftOperators<Byte,Int32,Byte>.UnsignedRightShift(Byte, Int32)	Shifts a value right by a given amount.
ISubtractionOperators<Byte,Byte,Byte>.CheckedSubtraction(Byte, Byte)	Subtracts two values to compute their difference.
ISubtractionOperators<Byte,Byte,Byte>.Subtraction(Byte, Byte)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Byte,Byte>.CheckedUnaryNegation(Byte)	Computes the checked unary negation of a value.
IUnaryNegationOperators<Byte,Byte>.UnaryNegation(Byte)	Computes the unary negation of a value.
IUnaryPlusOperators<Byte,Byte>.UnaryPlus(Byte)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- SByte
- IComparable
- IFormattable
- IConvertible
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of a Byte. This field is constant.

C#

```
public const byte MaxValue = 255;
```

Field Value

***Value* = 255**

Byte

Examples

The following example demonstrates how to use the MaxValue field to screen variable inputs for values that are outside the range of possible byte values.

C#

```
public void MinMaxFields(int numberToSet)
{
   if(numberToSet <= (int)Byte.MaxValue && numberToSet >=
(int)Byte.MinValue)
   {
      // You must explicitly convert an integer to a byte.
      MemberByte = (Byte)numberToSet;

      // Displays MemberByte using the ToString() method.
      Console.WriteLine("The MemberByte value is {0}",
MemberByte.ToString());
   }
   else
   {
      Console.WriteLine("The value {0} is outside of the range of possible
Byte values", numberToSet.ToString());
```

```
    }
}
```

Remarks

The value of this constant is 255 (hexadecimal 0xFF).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of a Byte. This field is constant.

C#

```
public const byte MinValue = 0;
```

Field Value

Value* = *0

Byte

Examples

The following example demonstrates how to use the MinValue field to screen variable inputs for values that are outside the range of possible byte values.

C#

```
public void MinMaxFields(int numberToSet)
{
   if(numberToSet <= (int)Byte.MaxValue && numberToSet >=
(int)Byte.MinValue)
   {
      // You must explicitly convert an integer to a byte.
      MemberByte = (Byte)numberToSet;

      // Displays MemberByte using the ToString() method.
      Console.WriteLine("The MemberByte value is {0}",
MemberByte.ToString());
   }
   else
   {
      Console.WriteLine("The value {0} is outside of the range of possible
Byte values", numberToSet.ToString());
```

```
    }
}
```

Remarks

The value of this constant is 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.Clamp(Byte, Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static byte Clamp (byte value, byte min, byte max);
```

Parameters

`value` Byte

The value to clamp.

`min` Byte

The inclusive minimum to which `value` should clamp.

`max` Byte

The inclusive maximum to which `value` should clamp.

Returns

Byte

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Byte and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Byte)	Compares this instance to a specified 8-bit unsigned integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(Byte)

Compares this instance to a specified 8-bit unsigned integer and returns an indication of their relative values.

C#

```
public int CompareTo (byte value);
```

Parameters

`value` Byte

An 8-bit unsigned integer to compare.

Returns

Int32

A signed integer that indicates the relative order of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and non-generic versions of the `CompareTo` method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");
```

```
    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
```

```
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                            resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
```

```
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Byte.CompareTo(Object) method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- IComparable

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed integer that indicates the relative order of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.

Return Value	Description
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Byte.

Examples

The following code example demonstrates the CompareTo method.

C#

```
public void Compare(Byte myByte)
{
   int myCompareResult;

   myCompareResult = MemberByte.CompareTo(myByte);

   if(myCompareResult > 0)
   {
      Console.WriteLine("{0} is less than the MemberByte value {1}",
myByte.ToString(), MemberByte.ToString());
   }
   else if(myCompareResult < 0)
   {
      Console.WriteLine("{0} is greater than the MemberByte value {1}",
myByte.ToString(), MemberByte.ToString());
   }
   else
   {
      Console.WriteLine("{0} is equal to the MemberByte value {1}",
myByte.ToString(), MemberByte.ToString());
   }
}
```

Remarks

`value` must be `null` or an instance of Byte; otherwise, an exception is thrown.

Any instance of Byte, regardless of its value, is considered greater than `null`.

This method is implemented to support the IComparable interface.

See also

- ArgumentException
- IComparable
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static byte CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Byte

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static byte CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Byte

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static byte CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Byte

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.DivRem(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (byte Quotient, byte Remainder) DivRem (byte left, byte
right);
```

Parameters

`left` Byte

The value which `right` divides.

`right` Byte

The value which divides `left`.

Returns

ValueTuple<Byte,Byte>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two instances of Byte represent the same value.

Overloads

Expand table

Equals(Byte)	Returns a value indicating whether this instance and a specified Byte object represent the same value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Examples

The following code example determines whether the first Byte value is equal to the second Byte value, and whether the first Byte value is equal to the boxed version of the second Byte value.

C#

```
// This code example demonstrates the System.Byte.Equals(Object) and
// System.Byte.Equals(Byte) methods.

using System;

class Sample
{
    public static void Main()
    {
    byte   byteVal1 = 0x7f;
    byte   byteVal2 = 127;
    object objectVal3 = byteVal2;
//
    Console.WriteLine("byteVal1 = {0}, byteVal2 = {1}, objectVal3 = {2}\n",
                      byteVal1, byteVal2, objectVal3);
    Console.WriteLine("byteVal1 equals byteVal2?: {0}",
```

```
byteVal1.Equals(byteVal2));
    Console.WriteLine("byteVal1 equals objectVal3?: {0}",
byteVal1.Equals(objectVal3));
    }
}

/*
This code example produces the following results:

byteVal1 = 127, byteVal2 = 127, objectVal3 = 127

byteVal1 equals byteVal2?: True
byteVal1 equals objectVal3?: True

*/
```

Equals(Byte)

Returns a value indicating whether this instance and a specified Byte object represent the same value.

```
C#

public bool Equals (byte obj);
```

Parameters

`obj` Byte

An object to compare to this instance.

Returns

Boolean

`true` if `obj` is equal to this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals(Object) because it does not have to convert the `obj` parameter to

an object.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare with this instance, or `null`.

Returns

Boolean

`true` if `obj` is an instance of Byte and equals the value of this instance; otherwise, `false`.

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A hash code for the current Byte.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetHashCode()

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Byte.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Byte.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Byte.IsEvenInteger(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IsOddInteger(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (byte value);
```

Parameters

value Byte

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IsPow2(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.LeadingZeroCount(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static byte LeadingZeroCount (byte value);
```

Parameters

`value` Byte

The value whose leading zeroes are to be counted.

Returns

Byte

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Log2(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static byte Log2 (byte value);
```

Parameters

`value` Byte

The value whose log2 is to be computed.

Returns

Byte

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Max(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static byte Max (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Min(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static byte Min (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its Byte equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its Byte equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its Byte equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its Byte equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Converts the string representation of a number to its Byte equivalent.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its Byte equivalent.

```C#
public static byte Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific information about the format of `s`. If `provider` is `null`, the thread current culture is used.

Returns

Byte

A byte value that is equivalent to the number contained in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Byte.MinValue or greater than Byte.MaxValue.

-or-

`s` includes non-zero, fractional digits.

ArgumentException
`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following code example parses string representations of `Byte` values with this overload of the Byte.Parse(String, NumberStyles, IFormatProvider) method.

C#

```
NumberStyles style;
CultureInfo culture;
string value;
byte number;

// Parse number with decimals.
// NumberStyles.Float includes NumberStyles.AllowDecimalPoint.
style = NumberStyles.Float;
culture = CultureInfo.CreateSpecificCulture("fr-FR");
value = "12,000";

number = Byte.Parse(value, style, culture);
Console.WriteLine("Converted '{0}' to {1}.", value, number);

culture = CultureInfo.CreateSpecificCulture("en-GB");
try
{
   number = Byte.Parse(value, style, culture);
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
}
catch (FormatException) {
   Console.WriteLine("Unable to parse '{0}'.", value); }

value = "12.000";
number = Byte.Parse(value, style, culture);
Console.WriteLine("Converted '{0}' to {1}.", value, number);
// The example displays the following output to the console:
//       Converted '12,000' to 12.
```

```
//       Unable to parse '12,000'.
//       Converted '12.000' to 12.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign]digits[.fractional_digits][e[sign]digits][ws]

Or, if the `style` parameter includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional positive sign. (The method throws an OverflowException if a negative sign is present in `s`.) The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.

Element	Description
e	The e or E character, which indicates that the value is represented in exponential notation. The s parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A positive sign can appear after *digits*.
NumberStyles.AllowParentheses	Although this flag is supported, the use of parentheses in `s` results in an OverflowException.
NumberStyles.AllowThousands	Although the group separator symbol can appear in `s`, it can be preceded by only one or more 0 digits.

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.AllowCurrencySymbol	The $ element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "F3" parses successfully, but "0xF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the thread current culture is used.

See also

- ToString()
- MaxValue
- MinValue
- NumberStyles
- NumberFormatInfo
- IFormatProvider
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent.

C#

```
public static byte Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the value to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific information about the format of `s`. If `provider` is `null`, the thread current culture is used.

Returns

Byte

A byte value that is equivalent to the number contained in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static byte Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Byte

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its Byte equivalent.

```
C#
public static byte Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert. The string is interpreted using the Integer style.

`provider` IFormatProvider

An object that supplies culture-specific parsing information about `s`. If `provider` is `null`, the thread current culture is used.

Returns

Byte

A byte value that is equivalent to the number contained in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example parses string representations of `Byte` values with the Parse method.

C#

```
string stringToConvert;
byte byteValue;

stringToConvert = " 214 ";
try {
   byteValue = Byte.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert,
byteValue);
}
catch (FormatException) {
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert); }
catch (OverflowException) {
   Console.WriteLine("'{0}' is greater than {1} or less than {2}.",
                     stringToConvert, Byte.MaxValue, Byte.MinValue); }

stringToConvert = " + 214 ";
try {
   byteValue = Byte.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert,
byteValue);
}
catch (FormatException) {
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert); }
catch (OverflowException) {
   Console.WriteLine("'{0}' is greater than {1} or less than {2}.",
                     stringToConvert, Byte.MaxValue, Byte.MinValue); }

stringToConvert = " +214 ";
try {
   byteValue = Byte.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert,
byteValue);
}
catch (FormatException) {
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert); }
catch (OverflowException) {
   Console.WriteLine("'{0}' is greater than {1} or less than {2}.",
                     stringToConvert, Byte.MaxValue, Byte.MinValue); }
// The example displays the following output to the console:
```

```
//       Converted ' 214 ' to 214.
//       Unable to parse ' + 214 '.
//       Converted ' +214 ' to 214.
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional positive sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces together with a leading sign are allowed. (If the sign is present, it must be a positive sign or the method throws an OverflowException.) To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Byte.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object supplied by `provider`. The `provider` parameter is an IFormatProvider implementation such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the thread current culture is used.

See also

- ToString()
- TryParse
- MaxValue
- MinValue
- NumberStyles

- NumberFormatInfo
- IFormatProvider
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its Byte equivalent.

C#

```
public static byte Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string that contains a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

Returns

Byte

A byte value that is equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Byte.MinValue or greater than Byte.MaxValue.

-or-

`s` includes non-zero, fractional digits.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example parses string representations of `Byte` values with the Byte.Parse(String, NumberStyles) method. The current culture for the example is en-US.

C#

```
string value;
NumberStyles style;
byte number;

// Parse value with no styles allowed.
style = NumberStyles.None;
value = " 241 ";
try
{
   number = Byte.Parse(value, style);
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
}
catch (FormatException) {
   Console.WriteLine("Unable to parse '{0}'.", value); }
```

```
// Parse value with trailing sign.
style = NumberStyles.Integer | NumberStyles.AllowTrailingSign;
value = " 163+";
number = Byte.Parse(value, style);
Console.WriteLine("Converted '{0}' to {1}.", value, number);

// Parse value with leading sign.
value = "   +253  ";
number = Byte.Parse(value, style);
Console.WriteLine("Converted '{0}' to {1}.", value, number);
// This example displays the following output to the console:
//       Unable to parse ' 241 '.
//       Converted ' 163+' to 163.
//       Converted '   +253  ' to 253.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign]digits[.fractional_digits][e[sign]digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of s if style includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional positive sign. (The method throws an OverflowException if a negative sign is present in `s`.) The sign can appear at the beginning of `s` if

Element	Description
	`style` includes the NumberStyles.AllowLeadingSign flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The e or E character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A positive sign can appear after *digits*.
NumberStyles.AllowParentheses	Although this flag is supported, the use of parentheses in `s` results in an OverflowException.
NumberStyles.AllowThousands	Although the group separator symbol can appear in `s`, it can be preceded by only one or more 0 digits.
NumberStyles.AllowCurrencySymbol	The *$* element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "F3" parses successfully, but "0xF3" does not. The only other flags that can be combined with it are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To use the formatting information of some other culture, call the Byte.Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- MaxValue
- MinValue
- NumberStyles
- NumberFormatInfo
- IFormatProvider
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#

public static byte Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Byte

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static byte Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Byte

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

Converts the string representation of a number to its Byte equivalent.

```
C#

public static byte Parse (string s);
```

Parameters

s String

A string that contains a number to convert. The string is interpreted using the Integer style.

Returns

Byte

A byte value that is equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example demonstrates how to convert a string value into a byte value using the Byte.Parse(String) method. The resulting byte value is then displayed to the console.

```
C#

string stringToConvert = " 162";
byte byteValue;
try
{
   byteValue = Byte.Parse(stringToConvert);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert,
```

```
byteValue);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert);
}
catch (OverflowException)
{
   Console.WriteLine("'{0}' is greater than {1} or less than {2}.",
                     stringToConvert, Byte.MaxValue, Byte.MinValue);
}
// The example displays the following output to the console:
//       Converted ' 162' to 162.
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional positive or negative sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces together with a leading sign are allowed. (If the sign is present, it must be a positive sign or the method throws an OverflowException.) To explicitly define the style elements that can be present in `s`, use either the Byte.Parse(String, NumberStyles) or the Byte.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other culture, use the Byte.Parse(String, NumberStyles, IFormatProvider) method.

See also

- ToString()
- MaxValue
- MinValue
- NumberStyles
- NumberFormatInfo
- Parsing Numeric Strings in .NET
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.PopCount(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static byte PopCount (byte value);
```

Parameters

`value` Byte

The value whose set bits are to be counted.

Returns

Byte

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.RotateLeft(Byte, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

C#

```
public static byte RotateLeft (byte value, int rotateAmount);
```

Parameters

`value` Byte

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Byte

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.RotateRight(Byte, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static byte RotateRight (byte value, int rotateAmount);
```

Parameters

`value` Byte

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Byte

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.Sign(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (byte value);
```

Parameters

`value` Byte

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current Byte object to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of the current Byte object to its equivalent string representation using the specified culture-specific formatting information.
ToString(String)	Converts the value of the current Byte object to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the value of the current Byte object to its equivalent string representation using the specified format and culture-specific formatting information.
ToString()	Converts the value of the current Byte object to its equivalent string representation.

ToString(IFormatProvider)

Converts the numeric value of the current Byte object to its equivalent string representation using the specified culture-specific formatting information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this object in the format specified by the `provider` parameter.

Implements

ToString(IFormatProvider)

Examples

The following example iterates an array of byte values and displays each of them to the console by calling the ToString(IFormatProvider) method with different format providers.

C#

```
byte[] bytes = {0, 1, 14, 168, 255};
CultureInfo[] providers = {new CultureInfo("en-us"),
                           new CultureInfo("fr-fr"),
                           new CultureInfo("de-de"),
                           new CultureInfo("es-es")};
foreach (byte byteValue in bytes)
{
   foreach (CultureInfo provider in providers)
      Console.Write("{0,3} ({1})      ",
                    byteValue.ToString(provider), provider.Name);

   Console.WriteLine();
}
// The example displays the following output to the console:
//      0 (en-US)        0 (fr-FR)        0 (de-DE)        0 (es-ES)
//      1 (en-US)        1 (fr-FR)        1 (de-DE)        1 (es-ES)
//     14 (en-US)       14 (fr-FR)       14 (de-DE)       14 (es-ES)
//    168 (en-US)      168 (fr-FR)      168 (de-DE)      168 (es-ES)
//    255 (en-US)      255 (fr-FR)      255 (de-DE)      255 (es-ES)
```

Remarks

The return value is formatted with the general numeric format specifier ("G").

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value is formatted using the NumberFormatInfo object for the thread current culture. For information about the thread current culture, see Thread.CurrentCulture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- NumberFormatInfo
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the value of the current Byte object to its equivalent string representation using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the current Byte object, formatted as specified by the `format` parameter.

Exceptions

FormatException

`format` includes an unsupported specifier. Supported format specifiers are listed in the Remarks section.

Examples

The following example initializes a Byte value and displays it to the console using each of the supported standard format strings and a custom format string. The example is run with en-US as the current culture.

```
C#
```

```
string[] formats = {"C3", "D4", "e1", "E2", "F1", "G", "N1",
                    "P0", "X4", "0000.0000"};
byte number = 240;
foreach (string format in formats)
   Console.WriteLine("'{0}' format specifier: {1}",
                     format, number.ToString(format));

// The example displays the following output to the console if the
// current culture is en-us:
//       'C3' format specifier: $240.000
//       'D4' format specifier: 0240
//       'e1' format specifier: 2.4e+002
//       'E2' format specifier: 2.40E+002
//       'F1' format specifier: 240.0
//       'G' format specifier: 240
//       'N1' format specifier: 240.0
//       'P0' format specifier: 24,000 %
//       'X4' format specifier: 00F0
//       '0000.0000' format specifier: 0240.0000
```

Remarks

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string (""), the return value is formatted with the general numeric format specifier ("G").

The return value of this function is formatted using the NumberFormatInfo object for the thread current culture. For information about the thread current culture, see Thread.CurrentCulture. To provide formatting information for cultures other than the current culture, call the Byte.ToString(String, IFormatProvider) method.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the value of the current Byte object to its equivalent string representation using the specified format and culture-specific formatting information.

```
C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A standard or custom numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the current Byte object, formatted as specified by the `format` and `provider` parameters.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` includes an unsupported specifier. Supported format specifiers are listed in the Remarks section.

Examples

The following example uses the standard "N" format string and four different CultureInfo objects to display the string representation of a byte value to the console.

C#

```
byte byteValue = 250;
CultureInfo[] providers = {new CultureInfo("en-us"),
                           new CultureInfo("fr-fr"),
                           new CultureInfo("es-es"),
                           new CultureInfo("de-de")};

foreach (CultureInfo provider in providers)
   Console.WriteLine("{0} ({1})",
                     byteValue.ToString("N2", provider), provider.Name);
// The example displays the following output to the console:
//       250.00 (en-US)
//       250,00 (fr-FR)
//       250,00 (es-ES)
//       250,00 (de-DE)
```

Remarks

The ToString(String, IFormatProvider) method formats a Byte value in a specified format of a specified culture. To format a number by using the default ("G") format of the current culture, call the ToString() method. To format a number by using a specified format of the current culture, call the ToString(String) method.

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string ("), the return value of this method is formatted with the general numeric format specifier ("G").

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-

specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value is formatted using the NumberFormatInfo object for the thread current culture. For information about the thread current culture, see Thread.CurrentCulture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- NumberFormatInfo
- IFormattable
- IFormatProvider
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the value of the current Byte object to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this object, which consists of a sequence of digits that range from 0 to 9 with no leading zeroes.

Examples

The following example displays an array of byte values. Note that the ToString() method is not called explicitly in the example. Instead, it is called implicitly, because of the use of the composite formatting feature, the F# example uses string interpolation.

C#

```
byte[] bytes = {0, 1, 14, 168, 255};
foreach (byte byteValue in bytes)
   Console.WriteLine(byteValue);
// The example displays the following output to the console if the
current
// culture is en-US:
//       0
//       1
//       14
```

```
//        168
//        255
```

Remarks

The return value is formatted with the general numeric format specifier ("G") and the NumberFormatInfo object for the thread current culture. To define the formatting of the Byte value's string representation, call the ToString method. To define both the format specifiers and culture used to create the string representation of a Byte value, call the ToString method.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

For information about the thread current culture, see Thread.CurrentCulture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Byte.TrailingZeroCount(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static byte TrailingZeroCount (byte value);
```

Parameters

`value` Byte

The value whose trailing zeroes are to be counted.

Returns

Byte

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current 8-bit unsigned integer instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current 8-bit unsigned integer instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the charactes that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the string representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Byte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Byte)	Tries to convert the span representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, Byte)	Tries to convert the string representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Byte)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Byte)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Byte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Byte)	Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Byte)	Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Byte)	Converts the string representation of a number in a specified style and culture-specific format to its Byte

	equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Byte)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out byte result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Byte

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Byte)

Tries to convert the span representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out byte result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`result` Byte

When this method returns, contains the Byte value equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Byte)

Tries to convert the string representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse (string? s, out byte result);
```

Parameters

s String

A string that contains a number to convert.

result Byte

When this method returns, contains the Byte value equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParse(String, Byte) method with a number of different string values.

C#

```
using System;

public class ByteConversion
{
   public static void Main()
   {
      string[] byteStrings = { null, string.Empty, "1024",
                               "100.1", "100", "+100", "-100",
                               "00000000000000100", "00,100",
                               "   20   ", "FF", "0x1F" };

      foreach (var byteString in byteStrings)
      {
          CallTryParse(byteString);
      }
   }

   private static void CallTryParse(string stringToConvert)
   {
```

```
        byte byteValue;
        bool success = Byte.TryParse(stringToConvert, out byteValue);
        if (success)
        {
           Console.WriteLine("Converted '{0}' to {1}",
                             stringToConvert, byteValue);
        }
        else
        {
           Console.WriteLine("Attempted conversion of '{0}' failed.",
                               stringToConvert);
        }
    }
}
// The example displays the following output to the console:
//       Attempted conversion of '' failed.
//       Attempted conversion of '' failed.
//       Attempted conversion of '1024' failed.
//       Attempted conversion of '100.1' failed.
//       Converted '100' to 100
//       Converted '+100' to 100
//       Attempted conversion of '-100' failed.
//       Converted '000000000000000100' to 100
//       Attempted conversion of '00,100' failed.
//       Converted '    20    ' to 20
//       Attempted conversion of 'FF' failed.
//       Attempted conversion of '0x1F' failed.
```

Remarks

The conversion fails and the method returns `false` if the `s` parameter is not in the correct format, if it is `null` or String.Empty, or if it represents a number less than MinValue or greater than MaxValue.

The Byte.TryParse(String, Byte) method is similar to the Byte.Parse(String) method, except that TryParse(String, Byte) does not throw an exception if the conversion fails.

The `s` parameter should be the string representation of a number in the following form:

```
[ws][sign]digits[ws]
```

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional positive sign, as specified by the NumberFormatInfo.PositiveSign property of the current culture.
digits	A sequence of decimal digits that range from 0 to 9.

The `s` parameter is interpreted using the Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces together with a leading sign are allowed. (If the sign is present, it must be a positive sign or the method throws an OverflowException.) To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Byte.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object for the current culture. For more information, see NumberFormatInfo.CurrentInfo.

This overload of the Byte.TryParse(String, Byte) method interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the Byte.TryParse(String, NumberStyles, IFormatProvider, Byte) overload.

See also

- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, Byte)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out byte result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Byte

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Byte)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
byte result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Byte

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Byte)

Tries to parse a span of UTF-8 characters into a value.

```C#
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
byte result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Byte

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Byte)

Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit unsigned integer equivalent.

```C#
```

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text, out byte
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Byte

When this method returns, contains the 8-bit unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Byte)

Converts the span representation of a number in a specified style and culture-specific format to its Byte equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
byte result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert. The span is interpreted using the Integer style.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. If `provider` is `null`, the thread current culture is used.

`result` Byte

When this method returns, contains the 8-bit unsigned integer value equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Byte.MinValue or greater than Byte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Byte)

Converts the string representation of a number in a specified style and culture-specific format to its Byte equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out byte result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. If `provider` is `null`, the thread current culture is used.

`result` Byte

When this method returns, contains the 8-bit unsigned integer value equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Byte.MinValue or greater than Byte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, Byte) method with a number of different string values.

C#

```
using System;
using System.Globalization;

public class ByteConversion2
{
   public static void Main()
   {
      string byteString;
      NumberStyles styles;

      byteString = "1024";
      styles = NumberStyles.Integer;
      CallTryParse(byteString, styles);

      byteString = "100.1";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(byteString, styles);

      byteString = "100.0";
      CallTryParse(byteString, styles);

      byteString = "+100";
      styles = NumberStyles.Integer | NumberStyles.AllowLeadingSign
               | NumberStyles.AllowTrailingSign;
      CallTryParse(byteString, styles);

      byteString = "-100";
      CallTryParse(byteString, styles);

      byteString = "000000000000000100";
      CallTryParse(byteString, styles);

      byteString = "00,100";
      styles = NumberStyles.Integer | NumberStyles.AllowThousands;
      CallTryParse(byteString, styles);

      byteString = "2E+3   ";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(byteString, styles);

      byteString = "FF";
```

```
      styles = NumberStyles.HexNumber;
      CallTryParse(byteString, styles);

      byteString = "0x1F";
      CallTryParse(byteString, styles);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      Byte byteValue;
      bool result = Byte.TryParse(stringToConvert, styles,
                                  null as IFormatProvider, out
byteValue);
      if (result)
         Console.WriteLine("Converted '{0}' to {1}",
                        stringToConvert, byteValue);
      else
         Console.WriteLine("Attempted conversion of '{0}' failed.",
                           stringToConvert.ToString());
   }
}
// The example displays the following output to the console:
//       Attempted conversion of '1024' failed.
//       Attempted conversion of '100.1' failed.
//       Converted '100.0' to 100
//       Converted '+100' to 100
//       Attempted conversion of '-100' failed.
//       Converted '000000000000000100' to 100
//       Converted '00,100' to 100
//       Attempted conversion of '2E+3   ' failed.
//       Converted 'FF' to 255
//       Attempted conversion of '0x1F' failed.
```

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object supplied by the `provider` parameter.

The style parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign]digits[.fractional_digits][e[sign]digits][ws]

Or, if the `style` parameter includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of s if style includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional positive sign. (The parse operation fails if a negative sign is present in `s`.) The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The e or E character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

Note

Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or the method returns `false`.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation. If `s` represents a number in exponential notation, it must represent an integer within the range of the Byte data type without a non-zero, fractional component.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A positive sign can appear after *digits*.
NumberStyles.AllowParentheses	Although this flag is supported, the method returns `false` if parentheses are present in `s`.
NumberStyles.AllowThousands	Although the group separator symbol can appear in `s`, it can be preceded by only one or more 0 digits.
NumberStyles.AllowCurrencySymbol	The *$* element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "F3" parses successfully, but "0xF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a CultureInfo object or a NumberFormatInfo object, whose GetFormat method returns a

NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `s`.

See also

- ToString()
- MaxValue
- MinValue
- NumberStyles
- NumberFormatInfo
- IFormatProvider
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To Boolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To Byte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, unchanged.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To Int64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.To SByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToSByte(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this Byte value.

provider IFormatProvider

An IFormatProvider implementation that supplies information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

The requested type conversion is not supported.

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt16(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt32(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Byte instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt64(Byte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IAdditionOperators<Byte,Byte,Byte>.Addition(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static byte IAdditionOperators<byte,byte,byte>.operator + (byte left, byte right);
```

Parameters

`left` Byte

The value to which `right` is added.

`right` Byte

The value that's added to `left`.

Returns

Byte

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IAdditionOperators<Byte,Byte,Byte>.CheckedAdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static byte IAdditionOperators<byte,byte,byte>.op_CheckedAddition (byte
left, byte right);
```

Parameters

`left` Byte

The value to which `right` is added.

`right` Byte

The value that's added to `left`.

Returns

Byte

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IAdditiveIdentity<Byte,Byte>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static byte
System.Numerics.IAdditiveIdentity<System.Byte,System.Byte>.AdditiveIdentity
{ get; }
```

Property Value

Byte

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.IBinaryInteger<Byte>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<byte>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.IBinaryInteger<Byte>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<byte>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Byte.IBinaryInteger<Byte>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<byte>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out byte value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Byte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Byte.IBinaryInteger<Byte>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<byte>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out byte value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Byte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Byte.IBinaryInteger<Byte>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

```
C#

bool IBinaryInteger<byte>.TryWriteBigEndian (Span<byte> destination, out int
bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBinaryInteger<Byte>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<byte>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBinaryNumber<Byte>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static byte System.Numerics.IBinaryNumber<System.Byte>.AllBitsSet { get; }
```

Property Value

Byte

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBitwise Operators<Byte,Byte,Byte>.Bitwise And(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static byte IBitwiseOperators<byte,byte,byte>.operator & (byte left, byte
right);
```

Parameters

`left` Byte

The value to `and` with `right`.

`right` Byte

The value to `and` with `left`.

Returns

Byte

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBitwiseOperators<Byte,Byte,Byte>.BitwiseOr(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static byte IBitwiseOperators<byte,byte,byte>.operator | (byte left, byte
right);
```

Parameters

`left` Byte

The value to `or` with `right`.

`right` Byte

The value to `or` with `left`.

Returns

Byte

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBitwiseOperators<Byte,Byte,Byte>.ExclusiveOr(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static byte IBitwiseOperators<byte,byte,byte>.operator ^ (byte left, byte
right);
```

Parameters

`left` Byte

The value to xor with `right`.

`right` Byte

The value to xor with `left`.

Returns

Byte

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IBitwise Operators<Byte,Byte,Byte>.Ones Complement(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static byte IBitwiseOperators<byte,byte,byte>.operator ~ (byte value);
```

Parameters

`value` Byte

The value for which to compute the ones-complement.

Returns

Byte

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IComparisonOperators<Byte,Byte,Boolean>.GreaterThanOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<byte,byte,bool>.operator > (byte left, byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IComparisonOperators<Byte,Byte,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<byte,byte,bool>.operator >= (byte left,
byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IComparisonOperators<Byte,Byte,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<byte,byte,bool>.operator < (byte left, byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IComparisonOperators<Byte,Byte,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<byte,byte,bool>.operator <= (byte left,
byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IDecrementOperators<Byte>.CheckedDecrement(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static byte IDecrementOperators<byte>.op_CheckedDecrement (byte value);
```

Parameters

`value` Byte

The value to decrement.

Returns

Byte

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IDecrementOperators<Byte>.Decrement(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static byte IDecrementOperators<byte>.operator -- (byte value);
```

Parameters

`value` Byte

The value to decrement.

Returns

Byte

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IDivisionOperators<Byte,Byte,Byte>.Division(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static byte IDivisionOperators<byte,byte,byte>.operator / (byte left, byte right);
```

Parameters

`left` Byte

The value that `right` divides.

`right` Byte

The value that divides `left`.

Returns

Byte

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IEqualityOperators<Byte,Byte,Boolean>.Equality(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<byte,byte,bool>.operator == (byte left, byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IEqualityOperators<Byte,Byte,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<byte,byte,bool>.operator != (byte left, byte right);
```

Parameters

`left` Byte

The value to compare with `right`.

`right` Byte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IIncrementOperators<Byte>.CheckedIncrement(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static byte IIncrementOperators<byte>.op_CheckedIncrement (byte value);
```

Parameters

`value` Byte

The value to increment.

Returns

Byte

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IIncrementOperators<Byte>.Increment(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static byte IIncrementOperators<byte>.operator ++ (byte value);
```

Parameters

`value` Byte

The value to increment.

Returns

Byte

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IMinMaxValue<Byte>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static byte System.Numerics.IMinMaxValue<System.Byte>.MaxValue { get; }
```

Property Value

Byte

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IMinMaxValue<Byte>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static byte System.Numerics.IMinMaxValue<System.Byte>.MinValue { get; }
```

Property Value

Byte

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IModulusOperators<Byte,Byte,Byte>.Modulus(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static byte IModulusOperators<byte,byte,byte>.operator % (byte left, byte
right);
```

Parameters

`left` Byte

The value that `right` divides.

`right` Byte

The value that divides `left`.

Returns

Byte

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IMultiplicativeIdentity<Byte,Byte>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static byte
System.Numerics.IMultiplicativeIdentity<System.Byte,System.Byte>.MultiplicativeIdentity { get; }
```

Property Value

Byte

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.IMultiplyOperators<Byte,Byte,Byte>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static byte IMultiplyOperators<byte,byte,byte>.op_CheckedMultiply (byte
left, byte right);
```

Parameters

`left` Byte

The value that `right` multiplies.

`right` Byte

The value that multiplies `left`.

Returns

Byte

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IMultiplyOperators<Byte,Byte,Byte>.Multiply(Byte, Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static byte IMultiplyOperators<byte,byte,byte>.operator * (byte left, byte
right);
```

Parameters

`left` Byte

The value that `right` multiplies.

`right` Byte

The value that multiplies `left`.

Returns

Byte

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumber<Byte>.CopySign(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static byte INumber<byte>.CopySign (byte value, byte sign);
```

Parameters

`value` Byte

The value whose magnitude is used in the result.

`sign` Byte

The value whose sign is used in the result.

Returns

Byte

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumber<Byte>.MaxNumber(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

C#

```
static byte INumber<byte>.MaxNumber (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumber<Byte>.MinNumber(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

C#

```
static byte INumber<byte>.MinNumber (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.Abs(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static byte INumberBase<byte>.Abs (byte value);
```

Parameters

`value` Byte

The value for which to get its absolute.

Returns

Byte

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsCanonical(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<byte>.IsCanonical (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsComplexNumber(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<byte>.IsComplexNumber (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsFinite(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<byte>.IsFinite (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsImaginaryNumber(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<byte>.IsImaginaryNumber (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsInfinity(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<byte>.IsInfinity (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsInteger(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<byte>.IsInteger (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsNaN(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<byte>.IsNaN (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsNegative(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<byte>.IsNegative (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsNegativeInfinity(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<byte>.IsNegativeInfinity (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsNormal(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<byte>.IsNormal (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsPositive(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<byte>.IsPositive (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsPositiveInfinity(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<byte>.IsPositiveInfinity (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsRealNumber(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<byte>.IsRealNumber (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsSubnormal(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<byte>.IsSubnormal (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.IsZero(Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<byte>.IsZero (byte value);
```

Parameters

`value` Byte

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.Max Magnitude(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static byte INumberBase<byte>.MaxMagnitude (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.Max MagnitudeNumber(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static byte INumberBase<byte>.MaxMagnitudeNumber (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.MinMagnitude(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
static byte INumberBase<byte>.MinMagnitude (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.MinMagnitudeNumber(Byte, Byte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#
static byte INumberBase<byte>.MinMagnitudeNumber (byte x, byte y);
```

Parameters

x Byte

The value to compare with `y`.

y Byte

The value to compare with `x`.

Returns

Byte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static byte System.Numerics.INumberBase<System.Byte>.One { get; }
```

Property Value

Byte

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.INumberBase<Byte>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Byte>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<byte>.TryConvertFromChecked<TOther> (TOther value,
out byte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Byte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.INumberBase<Byte>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<byte>.TryConvertFromSaturating<TOther> (TOther
value, out byte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Byte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.INumberBase<Byte>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<byte>.TryConvertFromTruncating<TOther> (TOther
value, out byte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Byte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Byte.INumberBase<Byte>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<byte>.TryConvertToChecked<TOther> (byte value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Byte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.INumberBase<Byte>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<byte>.TryConvertToSaturating<TOther> (byte value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Byte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<byte>.TryConvertToTruncating<TOther> (byte value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Byte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.INumberBase<Byte>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static byte System.Numerics.INumberBase<System.Byte>.Zero { get; }
```

Property Value

Byte

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Byte.IShift Operators<Byte,Int32,Byte>.Left Shift(Byte, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static byte IShiftOperators<byte,int,byte>.operator << (byte value, int
shiftAmount);
```

Parameters

`value` Byte

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Byte

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IShift Operators<Byte,Int32,Byte>.Right Shift(Byte, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static byte IShiftOperators<byte,int,byte>.operator >> (byte value, int
shiftAmount);
```

Parameters

`value` Byte

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Byte

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IShift Operators<Byte,Int32,Byte>.Unsigned RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static byte IShiftOperators<byte,int,byte>.op_UnsignedRightShift (byte
value, int shiftAmount);
```

Parameters

`value` Byte

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Byte

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.ISubtractionOperators<Byte,Byte,Byte>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static byte ISubtractionOperators<byte,byte,byte>.op_CheckedSubtraction
(byte left, byte right);
```

Parameters

`left` Byte

The value from which `right` is subtracted.

`right` Byte

The value that's subtracted from `left`.

Returns

Byte

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.ISubtraction Operators<Byte,Byte,Byte>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static byte ISubtractionOperators<byte,byte,byte>.operator - (byte left,
byte right);
```

Parameters

`left` Byte

The value from which `right` is subtracted.

`right` Byte

The value that's subtracted from `left`.

Returns

Byte

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IUnaryNegationOperators<Byte,Byte>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static byte IUnaryNegationOperators<byte,byte>.op_CheckedUnaryNegation (byte value);
```

Parameters

`value` Byte

The value for which to compute the unary negation.

Returns

Byte

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IUnaryNegationOperators<Byte,Byte>.UnaryNegation(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static byte IUnaryNegationOperators<byte,byte>.operator - (byte value);
```

Parameters

`value` Byte

The value for which to compute the unary negation.

Returns

Byte

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Byte.IUnaryPlusOperators<Byte,Byte>.UnaryPlus(Byte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static byte IUnaryPlusOperators<byte,byte>.operator + (byte value);
```

Parameters

`value` Byte

The value for which to compute the unary plus.

Returns

Byte

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

CannotUnloadAppDomainException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt to unload an application domain fails.

```
C#
public class CannotUnloadAppDomainException : SystemException
```

Inheritance Object → Exception → SystemException → CannotUnloadAppDomainException

Remarks

CannotUnloadAppDomainException is thrown when there is an attempt to unload the following:

- The default application domain, which must remain loaded during the lifetime of the application.
- An application domain with a running thread that cannot immediately stop execution.
- An application domain that has already been unloaded.

CannotUnloadAppDomainException uses the HRESULT COR_E_CANNOTUNLOADAPPDOMAIN, which has the value 0x80131015.

For a list of initial property values for an instance of CannotUnloadAppDomainException, see the CannotUnloadAppDomainException constructors.

Constructors

Expand table

CannotUnloadAppDomainException()	Initializes a new instance of the CannotUnloadAppDomainException class.
CannotUnloadAppDomain Exception(SerializationInfo, Streaming Context)	**Obsolete.** Initializes a new instance of the CannotUnloadAppDomainException class from serialized data.
CannotUnloadAppDomain Exception(String)	Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message.
CannotUnloadAppDomain Exception(String, Exception)	Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Object
- Exception
- AppDomain
- Thread
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CannotUnloadAppDomainException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the CannotUnloadAppDomainException class.

Overloads

Expand table

CannotUnloadAppDomainException()	Initializes a new instance of the CannotUnloadAppDomainException class.
CannotUnloadAppDomainException(String)	Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message.
CannotUnloadAppDomainException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the CannotUnloadAppDomainException class from serialized data.
CannotUnloadAppDomainException(String, Exception)	Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message and a reference to the inner exception that is the cause of this exception.

CannotUnloadAppDomainException()

Initializes a new instance of the CannotUnloadAppDomainException class.

```
C#
```

```
public CannotUnloadAppDomainException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempt to unload the AppDomain failed." This message takes into account the current system culture.

The following table shows the initial property values for an instance of CannotUnloadAppDomainException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CannotUnloadAppDomainException(String)

Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message.

C#

```
public CannotUnloadAppDomainException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of CannotUnloadAppDomainException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CannotUnloadAppDomainException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the CannotUnloadAppDomainException class from serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected CannotUnloadAppDomainException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CannotUnloadAppDomainException(String, Exception) *(Obsolete)*

Initializes a new instance of the CannotUnloadAppDomainException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public CannotUnloadAppDomainException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of CannotUnloadAppDomainException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a character as a UTF-16 code unit.

C#

```
public readonly struct Char : IComparable<char>, IConvertible,
IEquatable<char>, IParsable<char>, ISpanParsable<char>,
IUtf8SpanParsable<char>, System.Numerics.IAdditionOperators<char,char,char>,
System.Numerics.IAdditiveIdentity<char,char>,
System.Numerics.IBinaryInteger<char>, System.Numerics.IBinaryNumber<char>,
System.Numerics.IBitwiseOperators<char,char,char>,
System.Numerics.IComparisonOperators<char,char,bool>,
System.Numerics.IDecrementOperators<char>,
System.Numerics.IDivisionOperators<char,char,char>,
System.Numerics.IEqualityOperators<char,char,bool>,
System.Numerics.IIncrementOperators<char>,
System.Numerics.IMinMaxValue<char>,
System.Numerics.IModulusOperators<char,char,char>,
System.Numerics.IMultiplicativeIdentity<char,char>,
System.Numerics.IMultiplyOperators<char,char,char>,
System.Numerics.INumber<char>, System.Numerics.INumberBase<char>,
System.Numerics.IShiftOperators<char,int,char>,
System.Numerics.ISubtractionOperators<char,char,char>,
System.Numerics.IUnaryNegationOperators<char,char>,
System.Numerics.IUnaryPlusOperators<char,char>,
System.Numerics.IUnsignedNumber<char>
```

Inheritance Object → ValueType → Char

Implements IComparable , IComparable<Char> , IConvertible , IEquatable<Char> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Char> , IParsable<TSelf> , ISpanParsable<Char> , ISpanParsable<TSelf> , IAdditionOperators<Char,Char,Char> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Char,Char> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Char> ,

IBinaryNumber<Char> , IBinaryNumber<TSelf> ,
IBitwiseOperators<Char,Char,Char> , IBitwiseOperators<TSelf,TSelf,TSelf> ,
IComparisonOperators<Char,Char,Boolean> ,
IComparisonOperators<TSelf,TSelf,Boolean> ,
IDecrementOperators<Char> , IDecrementOperators<TSelf> ,
IDivisionOperators<Char,Char,Char> ,
IDivisionOperators<TSelf,TSelf,TSelf> ,
IEqualityOperators<Char,Char,Boolean> ,
IEqualityOperators<TSelf,TOther,TResult> ,
IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Char> ,
IIncrementOperators<TSelf> , IMinMaxValue<Char> ,
IModulusOperators<Char,Char,Char> ,
IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<Char,Char> ,
IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<Char,Char,Char> ,
IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Char> , INumber<TSelf> ,
INumberBase<Char> , INumberBase<TSelf> ,
IShiftOperators<Char,Int32,Char> , IShiftOperators<TSelf,Int32,TSelf> ,
ISubtractionOperators<Char,Char,Char> ,
ISubtractionOperators<TSelf,TSelf,TSelf> ,
IUnaryNegationOperators<Char,Char> ,
IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<Char,Char> ,
IUnaryPlusOperators<TSelf,TSelf> , IUnsignedNumber<Char> ,
IUtf8SpanFormattable , IUtf8SpanParsable<Char> ,
IUtf8SpanParsable<TSelf>

Examples

The following code example demonstrates some of the methods in Char.

C#

```
using System;

public class CharStructureSample
{
    public static void Main()
```

```
    {
        char chA = 'A';
        char ch1 = '1';
        string str = "test string";

        Console.WriteLine(chA.CompareTo('B'));          //-----------
Output: "-1" (meaning 'A' is 1 less than 'B')
        Console.WriteLine(chA.Equals('A'));             //-----------
Output: "True"
        Console.WriteLine(Char.GetNumericValue(ch1));   //-----------
Output: "1"
        Console.WriteLine(Char.IsControl('\t'));        //-----------
Output: "True"
        Console.WriteLine(Char.IsDigit(ch1));           //-----------
Output: "True"
        Console.WriteLine(Char.IsLetter(','));          //-----------
Output: "False"
        Console.WriteLine(Char.IsLower('u'));           //-----------
Output: "True"
        Console.WriteLine(Char.IsNumber(ch1));          //-----------
Output: "True"
        Console.WriteLine(Char.IsPunctuation('.'));     //-----------
Output: "True"
        Console.WriteLine(Char.IsSeparator(str, 4));    //-----------
Output: "True"
        Console.WriteLine(Char.IsSymbol('+'));          //-----------
Output: "True"
        Console.WriteLine(Char.IsWhiteSpace(str, 4));   //-----------
Output: "True"
        Console.WriteLine(Char.Parse("S"));             //-----------
Output: "S"
        Console.WriteLine(Char.ToLower('M'));           //-----------
Output: "m"
        Console.WriteLine('x'.ToString());              //-----------
Output: "x"
    }
}
```

Remarks

For more information about this API, see Supplemental API remarks for Char.

Fields

Expand table

MaxValue	Represents the largest possible value of a Char. This field is constant.
MinValue	Represents the smallest possible value of a Char. This field is constant.

Methods

Expand table

CompareTo(Char)	Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.
CompareTo(Object)	Compares this instance to a specified object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.
ConvertFromUtf32(Int32)	Converts the specified Unicode code point into a UTF-16 encoded string.
ConvertToUtf32(Char, Char)	Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.
ConvertToUtf32(String, Int32)	Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.
Equals(Char)	Returns a value that indicates whether this instance is equal to the specified Char object.
Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetNumericValue(Char)	Converts the specified numeric Unicode character to a double-precision floating point number.
GetNumericValue(String, Int32)	Converts the numeric Unicode character at the specified position in a specified string to a double-precision floating point number.
GetTypeCode()	Returns the TypeCode for value type Char.
GetUnicodeCategory(Char)	Categorizes a specified Unicode character into a group identified by one of the UnicodeCategory values.
GetUnicodeCategory(String, Int32)	Categorizes the character at the specified position in a specified string into a group identified by one of the UnicodeCategory values.
IsAscii(Char)	Returns `true` if `c` is an ASCII character ([U+0000..U+007F]).
IsAsciiDigit(Char)	Indicates whether a character is categorized as an ASCII digit.
IsAsciiHexDigit(Char)	Indicates whether a character is categorized as an ASCII hexademical digit.

IsAsciiHexDigitLower(Char)	Indicates whether a character is categorized as an ASCII lower-case hexademical digit.
IsAsciiHexDigitUpper(Char)	Indicates whether a character is categorized as an ASCII upper-case hexademical digit.
IsAsciiLetter(Char)	Indicates whether a character is categorized as an ASCII letter.
IsAsciiLetterLower(Char)	Indicates whether a character is categorized as a lowercase ASCII letter.
IsAsciiLetterOrDigit(Char)	Indicates whether a character is categorized as an ASCII letter or digit.
IsAsciiLetterUpper(Char)	Indicates whether a character is categorized as an uppercase ASCII letter.
IsBetween(Char, Char, Char)	Indicates whether a character is within the specified inclusive range.
IsControl(Char)	Indicates whether the specified Unicode character is categorized as a control character.
IsControl(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a control character.
IsDigit(Char)	Indicates whether the specified Unicode character is categorized as a decimal digit.
IsDigit(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a decimal digit.
IsHighSurrogate(Char)	Indicates whether the specified Char object is a high surrogate.
IsHighSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a high surrogate.
IsLetter(Char)	Indicates whether the specified Unicode character is categorized as a Unicode letter.
IsLetter(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a Unicode letter.
IsLetterOrDigit(Char)	Indicates whether the specified Unicode character is categorized as a letter or a decimal digit.
IsLetterOrDigit(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a letter or a decimal digit.
IsLower(Char)	Indicates whether the specified Unicode character is categorized as a lowercase letter.

IsLower(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a lowercase letter.
IsLowSurrogate(Char)	Indicates whether the specified Char object is a low surrogate.
IsLowSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a low surrogate.
IsNumber(Char)	Indicates whether the specified Unicode character is categorized as a number.
IsNumber(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a number.
IsPunctuation(Char)	Indicates whether the specified Unicode character is categorized as a punctuation mark.
IsPunctuation(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a punctuation mark.
IsSeparator(Char)	Indicates whether the specified Unicode character is categorized as a separator character.
IsSeparator(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a separator character.
IsSurrogate(Char)	Indicates whether the specified character has a surrogate code unit.
IsSurrogate(String, Int32)	Indicates whether the character at the specified position in a specified string has a surrogate code unit.
IsSurrogatePair(Char, Char)	Indicates whether the two specified Char objects form a surrogate pair.
IsSurrogatePair(String, Int32)	Indicates whether two adjacent Char objects at a specified position in a string form a surrogate pair.
IsSymbol(Char)	Indicates whether the specified Unicode character is categorized as a symbol character.
IsSymbol(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a symbol character.
IsUpper(Char)	Indicates whether the specified Unicode character is categorized as an uppercase letter.
IsUpper(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as an uppercase letter.
IsWhiteSpace(Char)	Indicates whether the specified Unicode character is categorized as white space.

IsWhiteSpace(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as white space.
Parse(String)	Converts the value of the specified string to its equivalent Unicode character.
ToLower(Char)	Converts the value of a Unicode character to its lowercase equivalent.
ToLower(Char, CultureInfo)	Converts the value of a specified Unicode character to its lowercase equivalent using specified culture-specific formatting information.
ToLowerInvariant(Char)	Converts the value of a Unicode character to its lowercase equivalent using the casing rules of the invariant culture.
ToString()	Converts the value of this instance to its equivalent string representation.
ToString(Char)	Converts the specified Unicode character to its equivalent string representation.
ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation using the specified culture-specific format information.
ToUpper(Char)	Converts the value of a Unicode character to its uppercase equivalent.
ToUpper(Char, CultureInfo)	Converts the value of a specified Unicode character to its uppercase equivalent using specified culture-specific formatting information.
ToUpperInvariant(Char)	Converts the value of a Unicode character to its uppercase equivalent using the casing rules of the invariant culture.
TryParse(String, Char)	Converts the value of the specified string to its equivalent Unicode character. A return code indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<Char,Char,Char>.Addition(Char, Char)	Adds two values together to compute their sum.
IAdditionOperators<Char,Char,Char>.CheckedAddition(Char, Char)	Adds two values together to compute their sum.
IAdditiveIdentity<Char,Char>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<Char>.GetByteCount()	Gets the number of bytes that will be written as part of

	TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Char>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Char>.LeadingZeroCount(Char)	Computes the number of leading zero bits in a value.
IBinaryInteger<Char>.PopCount(Char)	Computes the number of bits that are set in a value.
IBinaryInteger<Char>.RotateLeft(Char, Int32)	Rotates a value left by a given amount.
IBinaryInteger<Char>.RotateRight(Char, Int32)	Rotates a value right by a given amount.
IBinaryInteger<Char>.TrailingZeroCount(Char)	Computes the number of trailing zero bits in a value.
IBinaryInteger<Char>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Char)	
IBinaryInteger<Char>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Char)	
IBinaryInteger<Char>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Char>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Char>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBinaryNumber<Char>.IsPow2(Char)	Determines if a value is a power of two.
IBinaryNumber<Char>.Log2(Char)	Computes the log2 of a value.
IBitwiseOperators<Char,Char,Char>.BitwiseAnd(Char, Char)	Computes the bitwise-and of two values.
IBitwiseOperators<Char,Char,Char>.BitwiseOr(Char, Char)	Computes the bitwise-or of two values.
IBitwiseOperators<Char,Char,Char>.ExclusiveOr(Char, Char)	Computes the exclusive-or of two values.
IBitwiseOperators<Char,Char,Char>.Ones Complement(Char)	Computes the ones-complement representation of a given value.

IComparisonOperators<Char,Char,Boolean>.GreaterThan(Char, Char)	Compares two values to determine which is greater.
IComparisonOperators<Char,Char,Boolean>.GreaterThanOrEqual(Char, Char)	Compares two values to determine which is greater or equal.
IComparisonOperators<Char,Char,Boolean>.LessThan(Char, Char)	Compares two values to determine which is less.
IComparisonOperators<Char,Char,Boolean>.LessThanOrEqual(Char, Char)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToDouble(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).

IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Char>.CheckedDecrement(Char)	Decrements a value.
IDecrementOperators<Char>.Decrement(Char)	Decrements a value.
IDivisionOperators<Char,Char,Char>.Division(Char, Char)	Divides one value by another to compute their quotient.
IEqualityOperators<Char,Char,Boolean>.Equality(Char, Char)	Compares two values to determine equality.
IEqualityOperators<Char,Char,Boolean>.Inequality(Char, Char)	Compares two values to determine inequality.
IFormattable.ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
IIncrementOperators<Char>.CheckedIncrement(Char)	Increments a value.
IIncrementOperators<Char>.Increment(Char)	Increments a value.
IMinMaxValue<Char>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Char>.MinValue	Gets the minimum value of the current type.
IModulusOperators<Char,Char,Char>.Modulus(Char, Char)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Char,Char>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
IMultiplyOperators<Char,Char,Char>.Checked Multiply(Char, Char)	Multiplies two values together to compute their product.
IMultiplyOperators<Char,Char,Char>.Multiply(Char, Char)	Multiplies two values together to compute their product.
INumberBase<Char>.Abs(Char)	Computes the absolute of a value.
INumberBase<Char>.IsCanonical(Char)	Determines if a value is in its canonical representation.

INumberBase<Char>.IsComplexNumber(Char)	Determines if a value represents a complex number.
INumberBase<Char>.IsEvenInteger(Char)	Determines if a value represents an even integral number.
INumberBase<Char>.IsFinite(Char)	Determines if a value is finite.
INumberBase<Char>.IsImaginaryNumber(Char)	Determines if a value represents a pure imaginary number.
INumberBase<Char>.IsInfinity(Char)	Determines if a value is infinite.
INumberBase<Char>.IsInteger(Char)	Determines if a value represents an integral number.
INumberBase<Char>.IsNaN(Char)	Determines if a value is NaN.
INumberBase<Char>.IsNegative(Char)	Determines if a value is negative.
INumberBase<Char>.IsNegativeInfinity(Char)	Determines if a value is negative infinity.
INumberBase<Char>.IsNormal(Char)	Determines if a value is normal.
INumberBase<Char>.IsOddInteger(Char)	Determines if a value represents an odd integral number.
INumberBase<Char>.IsPositive(Char)	Determines if a value is positive.
INumberBase<Char>.IsPositiveInfinity(Char)	Determines if a value is positive infinity.
INumberBase<Char>.IsRealNumber(Char)	Determines if a value represents a real number.
INumberBase<Char>.IsSubnormal(Char)	Determines if a value is subnormal.
INumberBase<Char>.IsZero(Char)	Determines if a value is zero.
INumberBase<Char>.MaxMagnitude(Char, Char)	Compares two values to compute which is greater.
INumberBase<Char>.MaxMagnitudeNumber(Char, Char)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Char>.MinMagnitude(Char, Char)	Compares two values to compute which is lesser.
INumberBase<Char>.MinMagnitudeNumber(Char, Char)	Compares two values to compute which has the lesser magnitude and

	returning the other value if an input is `NaN`.
INumberBase<Char>.One	Gets the value `1` for the type.
INumberBase<Char>.Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
INumberBase<Char>.Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.
INumberBase<Char>.Radix	Gets the radix, or base, for the type.
INumberBase<Char>.TryConvertFromChecked<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertFromSaturating<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertFromTruncating<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertToChecked<TOther>(Char, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Char>.TryConvertToSaturating<TOther>(Char, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Char>.TryConvertToTruncating<TOther>(Char, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Char>.TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Char)	
INumberBase<Char>.TryParse(String, NumberStyles, IFormatProvider, Char)	
INumberBase<Char>.Zero	Gets the value `0` for the type.
IParsable<Char>.Parse(String, IFormatProvider)	Parses a string into a value.
IParsable<Char>.TryParse(String, IFormatProvider, Char)	
IShiftOperators<Char,Int32,Char>.LeftShift(Char, Int32)	Shifts a value left by a given amount.
IShiftOperators<Char,Int32,Char>.RightShift(Char, Int32)	Shifts a value right by a given amount.
IShiftOperators<Char,Int32,Char>.UnsignedRight	Shifts a value right by a given amount.

Shift(Char, Int32)	
ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.
ISpanParsable<Char>.Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
ISpanParsable<Char>.TryParse(ReadOnlySpan<Char>, IFormatProvider, Char)	
ISubtractionOperators<Char,Char,Char>.CheckedSubtraction(Char, Char)	Subtracts two values to compute their difference.
ISubtractionOperators<Char,Char,Char>.Subtraction(Char, Char)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Char,Char>.CheckedUnaryNegation(Char)	Computes the checked unary negation of a value.
IUnaryNegationOperators<Char,Char>.UnaryNegation(Char)	Computes the unary negation of a value.
IUnaryPlusOperators<Char,Char>.UnaryPlus(Char)	Computes the unary plus of a value.
IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- IComparable
- IConvertible
- String

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of a Char. This field is constant.

```
C#
public const char MaxValue;
```

Field Value

Char

Remarks

The value of this constant is hexadecimal 0xFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of a Char. This field is constant.

```
C#
public const char MinValue;
```

Field Value

Char

Remarks

The value of this constant is hexadecimal 0x00.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or value type, and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified object or value type.

Overloads

Expand table

CompareTo(Char)	Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.
CompareTo(Object)	Compares this instance to a specified object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.

CompareTo(Char)

Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.

C#

```
public int CompareTo (char value);
```

Parameters

`value` Char

A Char object to compare.

Returns

Int32

A signed number indicating the position of this instance in the sort order in relation to the `value` parameter.

Expand table

Return Value	Description
Less than zero	This instance precedes `value`.
Zero	This instance has the same position in the sort order as `value`.
Greater than zero	This instance follows `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and non-generic versions of the `CompareTo` method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
```

```
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
```

```
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:
```

```
The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Char.CompareTo method because it does not have to convert the `value` parameter to an object.

The comparison performed by this method is based on the encoded values of this instance and `value`, not their lexicographical characteristics.

If the CompareTo method parameter type has fewer bits (is narrower) than the instance type, some programming languages perform an implicit widening conversion that transforms the value of the parameter into a value with more bits.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Typically, implicit widening conversions are performed on numeric types. Consult your programming language's documentation to determine if its compiler performs

widening conversions. If the instance and parameter types mismatch and a suitable conversion is not available, the compiler generates instructions to box the parameter and invoke the CompareTo overload that takes an Object parameter.

See also

- IComparable
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare this instance to, or `null`.

Returns

Int32

A signed number indicating the position of this instance in the sort order in relation to the `value` parameter.

Expand table

Return Value	Description
Less than zero	This instance precedes `value`.
Zero	This instance has the same position in the sort order as `value`.
Greater than zero	This instance follows `value`. -or- `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Char object.

Examples

The following code example demonstrates CompareTo.

C#

```
using System;

public class CompareToSample {
    public static void Main() {
        char chA = 'A';
        char chB = 'B';

        Console.WriteLine(chA.CompareTo('A'));  // Output: "0" (meaning
they're equal)
        Console.WriteLine('b'.CompareTo(chB));  // Output: "32" (meaning
'b' is greater than 'B' by 32)
        Console.WriteLine(chA.CompareTo(chB));  // Output: "-1" (meaning
'A' is less than 'B' by 1)
```

```
    }
}
```

Remarks

The CompareTo method implements the IComparable interface.

The `value` parameter must be `null` or an instance of Char; otherwise, an exception is thrown.

The comparison performed by this method is based on the encoded values of this instance and `value`, not necessarily their lexicographical characteristics. Any instance of Char, regardless of its value, is considered greater than `null`.

See also

- IComparable
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ConvertFromUtf32(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified Unicode code point into a UTF-16 encoded string.

C#

```
public static string ConvertFromUtf32 (int utf32);
```

Parameters

`utf32` Int32

A 21-bit Unicode code point.

Returns

String

A string consisting of one Char object or a surrogate pair of Char objects equivalent to the code point specified by the `utf32` parameter.

Exceptions

ArgumentOutOfRangeException

`utf32` is not a valid 21-bit Unicode code point ranging from U+0 through U+10FFFF, excluding the surrogate pair range from U+D800 through U+DFFF.

Examples

The following code example demonstrates the ConvertToUtf32 and ConvertFromUtf32 methods.

C#

```
// This example demonstrates the Char.ConvertFromUtf32() method
//                            and Char.ConvertToUtf32() overloads.
```

```
using System;

class Sample
{
    public static void Main()
    {
    int letterA = 0x0041;  //U+00041 = LATIN CAPITAL LETTER A
    int music   = 0x1D161; //U+1D161 = MUSICAL SYMBOL SIXTEENTH NOTE
    string s1;
    string comment   = "Create a UTF-16 encoded string from a code point.";
    string comment1b = "Create a code point from a UTF-16 encoded string.";
    string comment2b = "Create a code point from a surrogate pair at a
certain position in a string.";
    string comment2c = "Create a code point from a high surrogate and a low
surrogate code point.";

//  Convert code point U+0041 to UTF-16. The UTF-16 equivalent of
//  U+0041 is a Char with hexadecimal value 0041.

    Console.WriteLine(comment);
    s1 = Char.ConvertFromUtf32(letterA);
    Console.Write("    1a) 0x{0:X} => ", letterA);
    Show(s1);
    Console.WriteLine();

//  Convert the lone UTF-16 character to a code point.

    Console.WriteLine(comment1b);
    letterA = Char.ConvertToUtf32(s1, 0);
    Console.Write("    1b) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", letterA);
    Console.WriteLine();

// ------------------------------------------------------------------

//  Convert the code point U+1D161 to UTF-16. The UTF-16 equivalent of
//  U+1D161 is a surrogate pair with hexadecimal values D834 and DD61.

    Console.WriteLine(comment);
    s1 = Char.ConvertFromUtf32(music);
    Console.Write("    2a) 0x{0:X} => ", music);
    Show(s1);
    Console.WriteLine();

//  Convert the surrogate pair in the string at index position
//  zero to a code point.

    Console.WriteLine(comment2b);
    music = Char.ConvertToUtf32(s1, 0);
    Console.Write("    2b) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", music);

//  Convert the high and low characters in the surrogate pair into a code
```

```
point.

    Console.WriteLine(comment2c);
    music = Char.ConvertToUtf32(s1[0], s1[1]);
    Console.Write("    2c) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", music);
    }

    private static void Show(string s)
    {
    for (int x = 0; x < s.Length; x++)
        {
        Console.Write("0x{0:X}{1}",
                      (int)s[x],
                      ((x == s.Length-1)? String.Empty : ", "));
        }
    }
}
/*
This example produces the following results:

Create a UTF-16 encoded string from a code point.
    1a) 0x41 => 0x41
Create a code point from a UTF-16 encoded string.
    1b) 0x41 => 0x41

Create a UTF-16 encoded string from a code point.
    2a) 0x1D161 => 0xD834, 0xDD61
Create a code point from a surrogate pair at a certain position in a string.
    2b) 0xD834, 0xDD61 => 0x1D161
Create a code point from a high surrogate and a low surrogate code point.
    2c) 0xD834, 0xDD61 => 0x1D161

*/
```

Remarks

Use this method to convert a 21-bit Unicode code point to a UTF-16 encoded string before testing the string with methods such as IsLowSurrogate(Char) and IsHighSurrogate(Char).

A valid code point outside the Basic Multilingual Plane (BMP) always yields a valid surrogate pair. However, a valid code point within the BMP might not yield a valid result according to the Unicode standard because no linguistic processing is used in the conversion. For that reason, use the System.Text.UTF32Encoding class to convert bulk UTF-32 data into bulk UTF-16 data.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Rune
- ConvertToUtf32(Char, Char)
- UTF32Encoding

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ConvertToUtf32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.

Overloads

Expand table

ConvertToUtf32(Char, Char)	Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.
ConvertToUtf32(String, Int32)	Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.

Examples

The following code example demonstrates the ConvertToUtf32 and ConvertFromUtf32 methods.

C#

```
// This example demonstrates the Char.ConvertFromUtf32() method
//                            and Char.ConvertToUtf32() overloads.
using System;

class Sample
{
    public static void Main()
    {
    int letterA = 0x0041;  //U+00041 = LATIN CAPITAL LETTER A
    int music   = 0x1D161; //U+1D161 = MUSICAL SYMBOL SIXTEENTH NOTE
    string s1;
    string comment   = "Create a UTF-16 encoded string from a code point.";
    string comment1b = "Create a code point from a UTF-16 encoded string.";
    string comment2b = "Create a code point from a surrogate pair at a
certain position in a string.";
    string comment2c = "Create a code point from a high surrogate and a low
surrogate code point.";
```

```
//  Convert code point U+0041 to UTF-16. The UTF-16 equivalent of
//  U+0041 is a Char with hexadecimal value 0041.

    Console.WriteLine(comment);
    s1 = Char.ConvertFromUtf32(letterA);
    Console.Write("    1a) 0x{0:X} => ", letterA);
    Show(s1);
    Console.WriteLine();

//  Convert the lone UTF-16 character to a code point.

    Console.WriteLine(comment1b);
    letterA = Char.ConvertToUtf32(s1, 0);
    Console.Write("    1b) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", letterA);
    Console.WriteLine();

// ------------------------------------------------------------------

//  Convert the code point U+1D161 to UTF-16. The UTF-16 equivalent of
//  U+1D161 is a surrogate pair with hexadecimal values D834 and DD61.

    Console.WriteLine(comment);
    s1 = Char.ConvertFromUtf32(music);
    Console.Write("    2a) 0x{0:X} => ", music);
    Show(s1);
    Console.WriteLine();

//  Convert the surrogate pair in the string at index position
//  zero to a code point.

    Console.WriteLine(comment2b);
    music = Char.ConvertToUtf32(s1, 0);
    Console.Write("    2b) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", music);

//  Convert the high and low characters in the surrogate pair into a code
point.

    Console.WriteLine(comment2c);
    music = Char.ConvertToUtf32(s1[0], s1[1]);
    Console.Write("    2c) ");
    Show(s1);
    Console.WriteLine(" => 0x{0:X}", music);
    }

    private static void Show(string s)
    {
    for (int x = 0; x < s.Length; x++)
        {
        Console.Write("0x{0:X}{1}",
                      (int)s[x],
```

```
                        ((x == s.Length-1)? String.Empty : ", "));
        }
    }
}
/*
This example produces the following results:

Create a UTF-16 encoded string from a code point.
    1a) 0x41 => 0x41
Create a code point from a UTF-16 encoded string.
    1b) 0x41 => 0x41

Create a UTF-16 encoded string from a code point.
    2a) 0x1D161 => 0xD834, 0xDD61
Create a code point from a surrogate pair at a certain position in a string.
    2b) 0xD834, 0xDD61 => 0x1D161
Create a code point from a high surrogate and a low surrogate code point.
    2c) 0xD834, 0xDD61 => 0x1D161

*/
```

ConvertToUtf32(Char, Char)

Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.

```
C#

public static int ConvertToUtf32 (char highSurrogate, char lowSurrogate);
```

Parameters

`highSurrogate` Char

A high surrogate code unit (that is, a code unit ranging from U+D800 through U+DBFF).

`lowSurrogate` Char

A low surrogate code unit (that is, a code unit ranging from U+DC00 through U+DFFF).

Returns

Int32

The 21-bit Unicode code point represented by the `highSurrogate` and `lowSurrogate` parameters.

Exceptions

ArgumentOutOfRangeException

`highSurrogate` is not in the range U+D800 through U+DBFF, or `lowSurrogate` is not in the range U+DC00 through U+DFFF.

Remarks

Use this method to convert a surrogate pair into a 21-bit Unicode code point. To convert UTF-16 data into UTF-32 data, use the System.Text.UTF32Encoding class.

Ordinarily, UTF-16 encoding represents a single Unicode character as a 16-bit code unit. However, it also supports surrogate pairs, which allow a single abstract character to be represented as two 16-bit code units. These two Char objects must have code units that range from U+D800 to U+DBFF for the first (high) surrogate and from U+DC00 to U+DFFF for the second (low) surrogate. Surrogate pairs are supported only by UTF-16 encoding. This method allows a character represented by a UTF-16 surrogate pair to be converted to a character using UTF-32 encoding.

See also

- Rune
- ConvertFromUtf32(Int32)
- UTF32Encoding

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ConvertToUtf32(String, Int32)

Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.

```
C#

public static int ConvertToUtf32 (string s, int index);
```

Parameters

`s` String

A string that contains a character or surrogate pair.

`index` Int32

The index position of the character or surrogate pair in `s`.

Returns

Int32

The 21-bit Unicode code point represented by the character or surrogate pair at the position in the `s` parameter specified by the `index` parameter.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is not a position within `s`.

ArgumentException

The specified index position contains a surrogate pair, and either the first character in the pair is not a valid high surrogate or the second character in the pair is not a valid low surrogate.

Remarks

Use this method to convert a character or surrogate pair into a 21-bit Unicode code point. To convert UTF-16 data into UTF-32 data, use the System.Text.UTF32Encoding class.

See also

- Rune
- ConvertFromUtf32(Int32)
- UTF32Encoding

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether this instance is equal to a specified object or Char value.

Overloads

Expand table

Equals(Char)	Returns a value that indicates whether this instance is equal to the specified Char object.
Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.

Equals(Char)

Returns a value that indicates whether this instance is equal to the specified Char object.

C#

```
public bool Equals (char obj);
```

Parameters

obj Char

An object to compare to this instance.

Returns

Boolean

`true` if the `obj` parameter equals the value of this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Char.Equals(Object) because it does not need to unbox the `obj` parameter.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value that indicates whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare with this instance or `null`.

Returns

Boolean

`true` if `obj` is an instance of Char and equals the value of this instance; otherwise, `false`.

Examples

The following code example demonstrates Equals.

C#

```
using System;

public class EqualsSample {
    public static void Main() {
        char chA = 'A';
        char chB = 'B';

        Console.WriteLine(chA.Equals('A'));     // Output: "True"
        Console.WriteLine('b'.Equals(chB));     // Output: "False"
    }
}
```

Remarks

The comparison performed by this method is based on the encoded values of this instance and `obj`, not necessarily their lexicographical characteristics.

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Char.GetNumericValue Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified numeric Unicode character to a double-precision floating-point number.

Overloads

Expand table

GetNumericValue(Char)	Converts the specified numeric Unicode character to a double-precision floating point number.
GetNumericValue(String, Int32)	Converts the numeric Unicode character at the specified position in a specified string to a double-precision floating point number.

GetNumericValue(Char)

Converts the specified numeric Unicode character to a double-precision floating point number.

C#

```
public static double GetNumericValue (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

Double

The numeric value of `c` if that character represents a number; otherwise, -1.0.

Examples

The following example demonstrates GetNumericValue.

```
C#

using System;

public class GetNumericValueSample {
    public static void Main() {
        string str = "input: 1";

        Console.WriteLine(Char.GetNumericValue('8'));       // Output:
"8"
        Console.WriteLine(Char.GetNumericValue(str, 7));    // Output:
"1"
    }
}
```

Remarks

The `c` parameter must be the Char representation of a numeric value. For example, if `c` is "5", the return value is 5. However, if `c` is "z", the return value is -1.0.

A character has an associated numeric value if and only if it is a member of one of the following UnicodeCategory categories: `DecimalDigitNumber`, `LetterNumber`, or `OtherNumber`.

The GetNumericValue method assumes that `c` corresponds to a single linguistic character and checks whether that character can be converted to a decimal digit. However, some numbers in the Unicode standard are represented by two Char objects that form a surrogate pair. For example, the Aegean numbering system consists of code points U+10107 through U+10133. The following example uses the ConvertFromUtf32 method to instantiate a string that represents AEGEAN NUMBER ONE. As the output from the example shows, the GetNumericValue(Char) method returns -1 if it is passed either a high surrogate or a low surrogate of this character.

```
C#

int utf32 = 0x10107;        // AEGEAN NUMBER ONE
string surrogate = Char.ConvertFromUtf32(utf32);
foreach (var ch in surrogate)
   Console.WriteLine("U+{0:X4}: {1}    ", Convert.ToUInt16(ch),
```

```
                                                Char.GetNumericValue(ch));

// The example displays the following output:
//       U+D800: -1
//       U+DD07: -1
```

See also

- GetNumericValue(Rune)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetNumericValue(String, Int32)

Converts the numeric Unicode character at the specified position in a specified string to a double-precision floating point number.

C#

```
public static double GetNumericValue (string s, int index);
```

Parameters

s String

A String.

index Int32

The character position in `s`.

Returns

Double

The numeric value of the character at position `index` in `s` if that character represents a number; otherwise, -1.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Examples

The following code example demonstrates GetNumericValue.

C#

```
using System;

public class GetNumericValueSample {
    public static void Main() {
        string str = "input: 1";

        Console.WriteLine(Char.GetNumericValue('8'));       // Output:
"8"
        Console.WriteLine(Char.GetNumericValue(str, 7));    // Output:
"1"
    }
}
```

Remarks

The `s` parameter must be the string representation of a numeric value. For example, if the character at position `index` in `s` is "5", the return value is 5. However, if the character at position `index` in `s` is "z", the return value is -1.

Character positions in a string are indexed starting from zero.

A character has an associated numeric value if and only if it is a member of one of the following UnicodeCategory categories: `DecimalDigitNumber`, `LetterNumber`, or `OtherNumber`.

If the Char object at position `index` is the first character of a valid surrogate pair, the GetNumericValue(String, Int32) method determines whether the surrogate pair forms a numeric digit. For example, the Aegean numbering system consists of code points U+10107 through U+10133. The following example uses the ConvertFromUtf32 method to instantiate a string that represents each Aegean number. As the output from the example shows, the GetNumericValue(String, Int32) method returns the correct numeric value if it is passed the high surrogate of an Aegean number. However, if it is passed the low surrogate, it considers only the low surrogate in isolation and returns -1.

C#

```
// Define a UTF32 value for each character in the
// Aegean numbering system.
for (int utf32 = 0x10107; utf32 <= 0x10133; utf32++) {
   string surrogate = Char.ConvertFromUtf32(utf32);
   for (int ctr = 0; ctr < surrogate.Length; ctr++)
      Console.Write("U+{0:X4} at position {1}: {2}     ",
                        Convert.ToUInt16(surrogate[ctr]), ctr,
                        Char.GetNumericValue(surrogate, ctr));

   Console.WriteLine();
}
// The example displays the following output:
//       U+D800 at position 0: 1     U+DD07 at position 1: -1
//       U+D800 at position 0: 2     U+DD08 at position 1: -1
//       U+D800 at position 0: 3     U+DD09 at position 1: -1
//       U+D800 at position 0: 4     U+DD0A at position 1: -1
//       U+D800 at position 0: 5     U+DD0B at position 1: -1
//       U+D800 at position 0: 6     U+DD0C at position 1: -1
//       U+D800 at position 0: 7     U+DD0D at position 1: -1
//       U+D800 at position 0: 8     U+DD0E at position 1: -1
//       U+D800 at position 0: 9     U+DD0F at position 1: -1
//       U+D800 at position 0: 10     U+DD10 at position 1: -1
//       U+D800 at position 0: 20     U+DD11 at position 1: -1
//       U+D800 at position 0: 30     U+DD12 at position 1: -1
//       U+D800 at position 0: 40     U+DD13 at position 1: -1
//       U+D800 at position 0: 50     U+DD14 at position 1: -1
//       U+D800 at position 0: 60     U+DD15 at position 1: -1
//       U+D800 at position 0: 70     U+DD16 at position 1: -1
//       U+D800 at position 0: 80     U+DD17 at position 1: -1
//       U+D800 at position 0: 90     U+DD18 at position 1: -1
//       U+D800 at position 0: 100     U+DD19 at position 1: -1
//       U+D800 at position 0: 200     U+DD1A at position 1: -1
//       U+D800 at position 0: 300     U+DD1B at position 1: -1
//       U+D800 at position 0: 400     U+DD1C at position 1: -1
//       U+D800 at position 0: 500     U+DD1D at position 1: -1
//       U+D800 at position 0: 600     U+DD1E at position 1: -1
//       U+D800 at position 0: 700     U+DD1F at position 1: -1
//       U+D800 at position 0: 800     U+DD20 at position 1: -1
//       U+D800 at position 0: 900     U+DD21 at position 1: -1
```

```
//       U+D800 at position 0: 1000     U+DD22 at position 1: -1
//       U+D800 at position 0: 2000     U+DD23 at position 1: -1
//       U+D800 at position 0: 3000     U+DD24 at position 1: -1
//       U+D800 at position 0: 4000     U+DD25 at position 1: -1
//       U+D800 at position 0: 5000     U+DD26 at position 1: -1
//       U+D800 at position 0: 6000     U+DD27 at position 1: -1
//       U+D800 at position 0: 7000     U+DD28 at position 1: -1
//       U+D800 at position 0: 8000     U+DD29 at position 1: -1
//       U+D800 at position 0: 9000     U+DD2A at position 1: -1
//       U+D800 at position 0: 10000     U+DD2B at position 1: -1
//       U+D800 at position 0: 20000     U+DD2C at position 1: -1
//       U+D800 at position 0: 30000     U+DD2D at position 1: -1
//       U+D800 at position 0: 40000     U+DD2E at position 1: -1
//       U+D800 at position 0: 50000     U+DD2F at position 1: -1
//       U+D800 at position 0: 60000     U+DD30 at position 1: -1
//       U+D800 at position 0: 70000     U+DD31 at position 1: -1
//       U+D800 at position 0: 80000     U+DD32 at position 1: -1
//       U+D800 at position 0: 90000     U+DD33 at position 1: -1
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Char.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Char.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.GetUnicodeCategory Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Categorizes a Unicode character into a group identified by one of the UnicodeCategory values.

Overloads

Expand table

GetUnicodeCategory(Char)	Categorizes a specified Unicode character into a group identified by one of the UnicodeCategory values.
GetUnicodeCategory(String, Int32)	Categorizes the character at the specified position in a specified string into a group identified by one of the UnicodeCategory values.

Examples

The following code example demonstrates GetUnicodeCategory.

C#

```
using System;

public class GetUnicodeCategorySample {
    public static void Main() {
        char ch2 = '2';
        string str = "Upper Case";

        Console.WriteLine(Char.GetUnicodeCategory('a'));        // Output:
"LowercaseLetter"
        Console.WriteLine(Char.GetUnicodeCategory(ch2));        // Output:
"DecimalDigitNumber"
        Console.WriteLine(Char.GetUnicodeCategory(str, 6));     // Output:
"UppercaseLetter"
    }
}
```

GetUnicodeCategory(Char)

Categorizes a specified Unicode character into a group identified by one of the UnicodeCategory values.

```
C#

public static System.Globalization.UnicodeCategory GetUnicodeCategory
(char c);
```

Parameters

c Char

The Unicode character to categorize.

Returns

UnicodeCategory

A UnicodeCategory value that identifies the group that contains `c`.

Remarks

The Char.GetUnicodeCategory method does not always return the same UnicodeCategory value as the CharUnicodeInfo.GetUnicodeCategory(Char) method when it is passed a particular character as a parameter. The CharUnicodeInfo.GetUnicodeCategory(Char) method is designed to reflect the current version of the Unicode standard. In contrast, although the Char.GetUnicodeCategory method usually reflects the current version of the Unicode standard, it may return a character's category based on a previous version of the standard or it may return a category that differs from the current standard in order to preserve backward compatibility. As a result, we recommend that you use the CharUnicodeInfo.GetUnicodeCategory(Char) method instead of Char.GetUnicodeCategory(Char).

Starting with .NET Framework 4.6.2, Unicode characters are classified based on The Unicode Standard, Version 8.0.0. In versions of the .NET Framework from the .NET Framework 4 to the .NET Framework 4.6.1, they are classified based on The Unicode Standard, Version 6.3.0.

See also

- GetUnicodeCategory(Rune)
- GetUnicodeCategory(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetUnicodeCategory(String, Int32)

Categorizes the character at the specified position in a specified string into a group identified by one of the UnicodeCategory values.

C#

```
public static System.Globalization.UnicodeCategory GetUnicodeCategory
(string s, int index);
```

Parameters

s String

A String.

index Int32

The character position in `s`.

Returns

UnicodeCategory

A UnicodeCategory enumerated constant that identifies the group that contains the character at position `index` in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

The Char.GetUnicodeCategory method does not always return the same UnicodeCategory value as the CharUnicodeInfo.GetUnicodeCategory(String, Int32) method when it is passed a particular character as a parameter. The CharUnicodeInfo.GetUnicodeCategory(String, Int32) method is designed to reflect the current version of the Unicode standard. In contrast, although the Char.GetUnicodeCategory method usually reflects the current version of the Unicode standard, it may return a character's category based on a previous version of the standard or it may return a category that differs from the current standard in order to preserve backward compatibility. As a result, we recommend that you use the CharUnicodeInfo.GetUnicodeCategory(Char) method instead of Char.GetUnicodeCategory(String, Int32).

Starting with .NET Framework 4.6.2, Unicode characters are classified based on The Unicode Standard, Version 8.0.0. In versions of the .NET Framework from the .NET Framework 4 to the .NET Framework 4.6.1, they are classified based on The Unicode Standard, Version 6.3.0.

See also

- GetUnicodeCategory(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsAscii(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns `true` if `c` is an ASCII character ([U+0000..U+007F]).

C#

```
public static bool IsAscii (char c);
```

Parameters

c Char

The character to analyze.

Returns

Boolean

`true` if `c` is an ASCII character; `false` otherwise.

Remarks

Per http://www.unicode.org/glossary/#ASCII, ASCII is only U+0000..U+007F.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiDigit(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII digit.

C#

```
public static bool IsAsciiDigit (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is an ASCII digit; otherwise, `false`.

Remarks

This method determines whether the character is in the range '0' through '9', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiHexDigit(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII hexademical digit.

C#

```
public static bool IsAsciiHexDigit (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is a hexademical digit; otherwise, `false`.

Remarks

This method determines whether the character is in the range '0' through '9', inclusive, 'A' through 'F', inclusive, or 'a' through 'f', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiHexDigitLower(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII lower-case hexademical digit.

C#

```
public static bool IsAsciiHexDigitLower (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is a lower-case hexademical digit; otherwise, `false`.

Remarks

This determines whether the character is in the range '0' through '9', inclusive, or 'a' through 'f', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiHexDigitUpper(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII upper-case hexademical digit.

C#

```
public static bool IsAsciiHexDigitUpper (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is a hexademical digit; otherwise, `false`.

Remarks

This determines whether the character is in the range '0' through '9', inclusive, or 'A' through 'F', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiLetter(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII letter.

C#

```
public static bool IsAsciiLetter (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is an ASCII letter; otherwise, `false`.

Remarks

This determines whether the character is in the range 'A' through 'Z', inclusive, or 'a' through 'z', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiLetterLower(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as a lowercase ASCII letter.

C#

```
public static bool IsAsciiLetterLower (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is a lowercase ASCII letter; otherwise, `false`.

Remarks

This determines whether the character is in the range 'a' through 'z', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiLetterOrDigit(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an ASCII letter or digit.

C#

```
public static bool IsAsciiLetterOrDigit (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is an ASCII letter or digit; otherwise, `false`.

Remarks

This determines whether the character is in the range 'A' through 'Z', inclusive, 'a' through 'z', inclusive, or '0' through '9', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsAsciiLetterUpper(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is categorized as an uppercase ASCII letter.

C#

```
public static bool IsAsciiLetterUpper (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if `c` is a lowercase ASCII letter; otherwise, `false`.

Remarks

This determines whether the character is in the range 'a' through 'z', inclusive.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsBetween(Char, Char, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character is within the specified inclusive range.

C#

```
public static bool IsBetween (char c, char minInclusive, char maxInclusive);
```

Parameters

`c` Char

The character to evaluate.

`minInclusive` Char

The lower bound, inclusive.

`maxInclusive` Char

The upper bound, inclusive.

Returns

Boolean

`true` if `c` is within the specified range; otherwise, `false`.

Remarks

The method does not validate that `maxInclusive` is greater than or equal to `minInclusive`. If `maxInclusive` is less than `minInclusive`, the behavior is undefined.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsControl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified Unicode character is categorized as a control character.

Overloads

Expand table

IsControl(Char)	Indicates whether the specified Unicode character is categorized as a control character.
IsControl(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a control character.

Remarks

Control characters are formatting and other non-printing characters, such as ACK, BEL, CR, FF, LF, and VT. The Unicode standard assigns code points from \U0000 to \U001F, \U007F, and from \U0080 to \U009F to control characters. According to the Unicode standard, these values are to be interpreted as control characters unless their use is otherwise defined by an application. Valid control characters are members of the UnicodeCategory.Control category.

IsControl(Char)

Indicates whether the specified Unicode character is categorized as a control character.

C#

```
public static bool IsControl (char c);
```

Parameters

`c` Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a control character; otherwise, `false`.

Examples

The following example lists the Unicode code point of each of the control characters.

C#

```
using System;

public class ControlChars
{
   public static void Main()
   {
      int charsWritten = 0;

      for (int ctr = 0x00; ctr <= 0xFFFF; ctr++)
      {
         char ch = Convert.ToChar(ctr);
         if (char.IsControl(ch))
         {
            Console.Write(@"\U{0:X4}    ", ctr);
            charsWritten++;
            if (charsWritten % 6 == 0)
               Console.WriteLine();
         }
      }
   }
}
// The example displays the following output to the console:
//       \U0000    \U0001    \U0002    \U0003    \U0004    \U0005
//       \U0006    \U0007    \U0008    \U0009    \U000A    \U000B
//       \U000C    \U000D    \U000E    \U000F    \U0010    \U0011
//       \U0012    \U0013    \U0014    \U0015    \U0016    \U0017
//       \U0018    \U0019    \U001A    \U001B    \U001C    \U001D
//       \U001E    \U001F    \U007F    \U0080    \U0081    \U0082
//       \U0083    \U0084    \U0085    \U0086    \U0087    \U0088
//       \U0089    \U008A    \U008B    \U008C    \U008D    \U008E
//       \U008F    \U0090    \U0091    \U0092    \U0093    \U0094
//       \U0095    \U0096    \U0097    \U0098    \U0099    \U009A
//       \U009B    \U009C    \U009D    \U009E    \U009F
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsControl(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a control character.

C#

```
public static bool IsControl (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a control character; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Examples

The following example enumerates the characters in a string and determines whether any are control characters.

C#

```
using System;

public class ControlChar
{
   public static void Main()
   {
      string sentence = "This is a " + Environment.NewLine + "two-line
sentence.";
      for (int ctr = 0; ctr < sentence.Length; ctr++)
      {
         if (Char.IsControl(sentence, ctr))
           Console.WriteLine("Control character \\U{0} found in position
{1}.",
              Convert.ToInt32(sentence[ctr]).ToString("X4"), ctr);
      }
   }
}
// The example displays the following output to the console:
//       Control character \U000D found in position 10.
//       Control character \U000A found in position 11.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsDigit Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a decimal digit.

Overloads

Expand table

IsDigit(Char)	Indicates whether the specified Unicode character is categorized as a decimal digit.
IsDigit(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a decimal digit.

Examples

The following code example demonstrates IsDigit.

C#

```
using System;

public class IsDigitSample {
    public static void Main() {
        char ch = '8';

        Console.WriteLine(Char.IsDigit(ch));                    // Output:
"True"
        Console.WriteLine(Char.IsDigit("sample string", 7));    // Output:
"False"
    }
}
```

Remarks

Valid digits are members of the UnicodeCategory.DecimalDigitNumber category.

IsDigit(Char)

Indicates whether the specified Unicode character is categorized as a decimal digit.

```
C#
public static bool IsDigit (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a decimal digit; otherwise, `false`.

Remarks

This method determines whether a Char is a radix-10 digit. This contrasts with IsNumber, which determines whether a `Char` is of any numeric Unicode category. Numbers include characters such as fractions, subscripts, superscripts, Roman numerals, currency numerators, encircled numbers, and script-specific digits.

Valid digits are members of the UnicodeCategory.DecimalDigitNumber category.

See also

- IsDigit(Rune)
- Boolean
- IsNumber(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsDigit(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a decimal digit.

C#

```
public static bool IsDigit (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a decimal digit; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

This method determines whether a Char is a radix-10 digit. This contrasts with IsNumber, which determines whether a Char is of any numeric Unicode category. Numbers include characters such as fractions, subscripts, superscripts, Roman numerals, currency numerators, encircled numbers, and script-specific digits.

Character positions in a string are indexed starting from zero.

Valid digits are members of the UnicodeCategory.DecimalDigitNumber category.

See also

- Boolean
- IsNumber(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsHighSurrogate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the specified Char object is a high surrogate.

Overloads

Expand table

IsHighSurrogate(Char)	Indicates whether the specified Char object is a high surrogate.
IsHighSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a high surrogate.

Examples

The following code example demonstrates the IsHighSurrogate, IsLowSurrogate, and IsSurrogatePair methods.

C#

```
// This example demonstrates the Char.IsLowSurrogate() method
//                                     IsHighSurrogate() method
//                                     IsSurrogatePair() method
using System;

class Sample
{
    public static void Main()
    {
    char cHigh = '\uD800';
    char cLow  = '\uDC00';
    string s1  = new String(new char[] {'a', '\uD800', '\uDC00', 'z'});
    string divider = String.Concat( Environment.NewLine, new String('-',
70),
                                    Environment.NewLine);

    Console.WriteLine();
    Console.WriteLine("Hexadecimal code point of the character, cHigh:
```

```
{0:X4}", (int)cHigh);
    Console.WriteLine("Hexadecimal code point of the character, cLow:
{0:X4}", (int)cLow);
    Console.WriteLine();
    Console.WriteLine("Characters in string, s1: 'a', high surrogate, low
surrogate, 'z'");
    Console.WriteLine("Hexadecimal code points of the characters in string,
s1: ");
    for(int i = 0; i < s1.Length; i++)
        {
        Console.WriteLine("s1[{0}] = {1:X4} ", i, (int)s1[i]);
        }
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a high
surrogate?");
    Console.WriteLine("A1) cLow?  - {0}", Char.IsHighSurrogate(cLow));
    Console.WriteLine("A2) cHigh? - {0}", Char.IsHighSurrogate(cHigh));
    Console.WriteLine("A3) s1[0]? - {0}", Char.IsHighSurrogate(s1, 0));
    Console.WriteLine("A4) s1[1]? - {0}", Char.IsHighSurrogate(s1, 1));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a low
surrogate?");
    Console.WriteLine("B1) cLow?  - {0}", Char.IsLowSurrogate(cLow));
    Console.WriteLine("B2) cHigh? - {0}", Char.IsLowSurrogate(cHigh));
    Console.WriteLine("B3) s1[0]? - {0}", Char.IsLowSurrogate(s1, 0));
    Console.WriteLine("B4) s1[2]? - {0}", Char.IsLowSurrogate(s1, 2));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following pairs of characters a
surrogate pair?");
    Console.WriteLine("C1) cHigh and cLow?  - {0}",
Char.IsSurrogatePair(cHigh, cLow));
    Console.WriteLine("C2) s1[0] and s1[1]? - {0}", Char.IsSurrogatePair(s1,
0));
    Console.WriteLine("C3) s1[1] and s1[2]? - {0}", Char.IsSurrogatePair(s1,
1));
    Console.WriteLine("C4) s1[2] and s1[3]? - {0}", Char.IsSurrogatePair(s1,
2));
    Console.WriteLine(divider);
    }
}
/*
This example produces the following results:

Hexadecimal code point of the character, cHigh: D800
Hexadecimal code point of the character, cLow:  DC00

Characters in string, s1: 'a', high surrogate, low surrogate, 'z'
Hexadecimal code points of the characters in string, s1:
s1[0] = 0061
s1[1] = D800
s1[2] = DC00
s1[3] = 007A
```

```
-----------------------------------------------------------------------

Is each of the following characters a high surrogate?
A1) cLow?  - False
A2) cHigh? - True
A3) s1[0]? - False
A4) s1[1]? - True

-----------------------------------------------------------------------

Is each of the following characters a low surrogate?
B1) cLow?  - True
B2) cHigh? - False
B3) s1[0]? - False
B4) s1[2]? - True

-----------------------------------------------------------------------

Is each of the following pairs of characters a surrogate pair?
C1) cHigh and cLow?  - True
C2) s1[0] and s1[1]? - False
C3) s1[1] and s1[2]? - True
C4) s1[2] and s1[3]? - False

-----------------------------------------------------------------------

*/
```

IsHighSurrogate(Char)

Indicates whether the specified Char object is a high surrogate.

C#

```
public static bool IsHighSurrogate (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if the numeric value of the `c` parameter ranges from U+D800 through U+DBFF; otherwise, `false`.

Remarks

In addition to representing single characters using a 16-bit code point, UTF-16 encoding allows abstract characters to be represented using two 16-bit code points, which is known as a surrogate pair. The first element in this pair is the high surrogate. Its code point can range from U+D800 to U+DBFF. An individual surrogate has no interpretation of its own; it is meaningful only when used as part of a surrogate pair.

See also

- Rune

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsHighSurrogate(String, Int32)

Indicates whether the Char object at the specified position in a string is a high surrogate.

C#

```
public static bool IsHighSurrogate (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the numeric value of the specified character in the `s` parameter ranges from U+D800 through U+DBFF; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is not a position within `s`.

Remarks

The `index` parameter is zero-based.

In addition to representing single characters using a 16-bit code point, UTF-16 encoding allows abstract characters to be represented using two 16-bit code points, which is known as a surrogate pair. The first element in this pair is the high surrogate. Its code point can range from U+D800 to U+DBFF. An individual surrogate has no interpretation of its own; it is meaningful only when used as part of a surrogate pair.

See also

- Rune

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsLetter Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a Unicode letter.

Overloads

Expand table

IsLetter(Char)	Indicates whether the specified Unicode character is categorized as a Unicode letter.
IsLetter(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a Unicode letter.

Examples

The following code example demonstrates IsLetter.

C#

```
using System;

public class IsLetterSample {
    public static void Main() {
        char ch = '8';

        Console.WriteLine(Char.IsLetter(ch));                    // False
        Console.WriteLine(Char.IsLetter("sample string", 7));    // True
    }
}
```

Remarks

This method determines whether a Char is a member of any category of Unicode letter. Unicode letters include the following:

- Uppercase letters, such as U+0041 (LATIN CAPITAL LETTER A) through U+005A (LATIN CAPITAL LETTER Z), or U+0400 (CYRILLIC CAPITAL LETTER IE WITH GRAVE) through U+042F (CYRILLIC CAPITAL LETTER YA). These characters are members of the UnicodeCategory.UppercaseLetter category.
- Lowercase letters, such as U+0061 (LATIN SMALL LETTER A) through U+007A (LATIN SMALL LETTER Z), or U+03AC (GREEK SMALL LETTER ALPHA WITH TONOS) through U+03CE (GREEK SMALL LETTER OMEGA WITH TONOS). These characters are members of the UnicodeCategory.LowercaseLetter category.
- Title case letters, such as U+01C5 (LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON) or U+1FFC (GREEK CAPITAL LETTER OMEGA WITH PROSGEGRAMMENI). These characters are members of the UnicodeCategory.TitlecaseLetter category.
- Modifiers, such as U+02B0 (MODIFIER LETTER SMALL H) through U+02C1 (MODIFIER LETTER REVERSED GLOTTAL STOP), or U+1D2C (MODIFIER LETTER CAPITAL A) through U+1D61 (MODIFIER LETTER SMALL CHI). These characters are members of the UnicodeCategory.ModifierLetter category.
- Other letters, such as U+05D0 (HEBREW LETTER ALEF) through U+05EA (HEBREW LETTER TAV), U+0621 (ARABIC LETTER HAMZA) through U+063A (ARABIC LETTER GHAIN), or U+4E00 (<CJK Ideograph, First>) through U+9FC3 (<CJK Ideograph, Last>). These characters are members of the UnicodeCategory.OtherLetter category.

IsLetter(Char)

Indicates whether the specified Unicode character is categorized as a Unicode letter.

C#

```
public static bool IsLetter (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a letter; otherwise, `false`.

Remarks

This method determines whether a Char is a member of any category of Unicode letter. Unicode letters include the following:

- Uppercase letters, such as U+0041 (LATIN CAPITAL LETTER A) through U+005A (LATIN CAPITAL LETTER Z), or U+0400 (CYRILLIC CAPITAL LETTER IE WITH GRAVE) through U+042F (CYRILLIC CAPITAL LETTER YA). These characters are members of the UnicodeCategory.UppercaseLetter category.
- Lowercase letters, such as U+0061 (LATIN SMALL LETTER A) through U+007A (LATIN SMALL LETTER Z), or U+03AC (GREEK SMALL LETTER ALPHA WITH TONOS) through U+03CE (GREEK SMALL LETTER OMEGA WITH TONOS). These characters are members of the UnicodeCategory.LowercaseLetter category.
- Title case letters, such as U+01C5 (LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON) or U+1FFC (GREEK CAPITAL LETTER OMEGA WITH PROSGEGRAMMENI). These characters are members of the UnicodeCategory.TitlecaseLetter category.
- Modifiers, such as U+02B0 (MODIFIER LETTER SMALL H) through U+02C1 (MODIFIER LETTER REVERSED GLOTTAL STOP), or U+1D2C (MODIFIER LETTER CAPITAL A) through U+1D61 (MODIFIER LETTER SMALL CHI). These characters are members of the UnicodeCategory.ModifierLetter category.
- Other letters, such as U+05D0 (HEBREW LETTER ALEF) through U+05EA (HEBREW LETTER TAV), U+0621 (ARABIC LETTER HAMZA) through U+063A (ARABIC LETTER GHAIN), or U+4E00 (<CJK Ideograph, First>) through U+9FC3 (<CJK Ideograph, Last>). These characters are members of the UnicodeCategory.OtherLetter category.

See also

- IsLetter(Rune)
- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsLetter(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a Unicode letter.

```
C#

public static bool IsLetter (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a letter; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

This method determines whether the character at a specified index position in a string is a member of any category of Unicode letter. Unicode letters include the following:

- Uppercase letters, such as U+0041 (LATIN CAPITAL LETTER A) through U+005A (LATIN CAPITAL LETTER Z), or U+0400 (CYRILLIC CAPITAL LETTER IE WITH GRAVE) through U+042F (CYRILLIC CAPITAL LETTER YA). These characters are members of the UnicodeCategory.UppercaseLetter category.
- Lowercase letters, such as U+0061 (LATIN SMALL LETTER A) through U+007A (LATIN SMALL LETTER Z), or U+03AC (GREEK SMALL LETTER ALPHA WITH TONOS) through U+03CE (GREEK SMALL LETTER OMEGA WITH TONOS). These characters are members of the UnicodeCategory.LowercaseLetter category.
- Title case letters, such as U+01C5 (LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON) or U+1FFC (GREEK CAPITAL LETTER OMEGA WITH PROSGEGRAMMENI). These characters are members of the UnicodeCategory.TitlecaseLetter category.
- Modifiers, such as U+02B0 (MODIFIER LETTER SMALL H) through U+02C1 (MODIFIER LETTER REVERSED GLOTTAL STOP), or U+1D2C (MODIFIER LETTER CAPITAL A) through U+1D61 (MODIFIER LETTER SMALL CHI). These characters are members of the UnicodeCategory.ModifierLetter category.
- Other letters, such as U+05D0 (HEBREW LETTER ALEF) through U+05EA (HEBREW LETTER TAV), U+0621 (ARABIC LETTER HAMZA) through U+063A (ARABIC LETTER GHAIN), or U+4E00 (<CJK Ideograph, First>) through U+9FC3 (<CJK Ideograph, Last>). These characters are members of the UnicodeCategory.OtherLetter category.

See also

- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsLetterOrDigit Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a letter or a decimal digit.

Overloads

Expand table

IsLetterOrDigit(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a letter or a decimal digit.
IsLetterOrDigit(Char)	Indicates whether the specified Unicode character is categorized as a letter or a decimal digit.

Examples

The following code example demonstrates IsLetterOrDigit.

C#

```
using System;

public class IsLetterOrDigitSample {
    public static void Main() {
        string str = "newline:\n";

        Console.WriteLine(Char.IsLetterOrDigit('8'));       // Output:
"True"
        Console.WriteLine(Char.IsLetterOrDigit(str, 8));    // Output:
"False", because it's a newline
    }
}
```

Remarks

Valid letters and decimal digits are members of the following categories in UnicodeCategory: `UppercaseLetter`, `LowercaseLetter`, `TitlecaseLetter`, `ModifierLetter`, `OtherLetter`, or `DecimalDigitNumber`.

IsLetterOrDigit(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a letter or a decimal digit.

```
C#
public static bool IsLetterOrDigit (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a letter or a decimal digit; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

Valid letters and decimal digits are members of the following categories in UnicodeCategory: `UppercaseLetter`, `LowercaseLetter`, `TitlecaseLetter`, `ModifierLetter`, `OtherLetter`, or `DecimalDigitNumber`.

See also

- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsLetterOrDigit(Char)

Indicates whether the specified Unicode character is categorized as a letter or a decimal digit.

C#

```
public static bool IsLetterOrDigit (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a letter or a decimal digit; otherwise, `false`.

Remarks

Valid letters and decimal digits are members of the following categories in UnicodeCategory: `UppercaseLetter`, `LowercaseLetter`, `TitlecaseLetter`, `ModifierLetter`, `OtherLetter`, or `DecimalDigitNumber`.

See also

- IsLetterOrDigit(Rune)
- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsLower Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a lowercase letter.

Overloads

Expand table

IsLower(Char)	Indicates whether the specified Unicode character is categorized as a lowercase letter.
IsLower(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a lowercase letter.

Examples

The following code example demonstrates IsLower.

C#

```
using System;

public class IsLowerSample {
    public static void Main() {
        char ch = 'a';

        Console.WriteLine(Char.IsLower(ch));                // Output:
"True"
        Console.WriteLine(Char.IsLower("upperCase", 5));    // Output:
"False"
    }
}
```

IsLower(Char)

Indicates whether the specified Unicode character is categorized as a lowercase letter.

```
C#

public static bool IsLower (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a lowercase letter; otherwise, `false`.

Remarks

Valid lowercase letters are members of the following category in UnicodeCategory: `LowercaseLetter`.

See also

- IsLower(Rune)
- IsUpper(Rune)
- Boolean
- IsUpper(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

IsLower(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a lowercase letter.

```
C#
public static bool IsLower (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a lowercase letter; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

Valid lowercase letters are members of the following category in UnicodeCategory: `LowercaseLetter`.

See also

- Boolean
- String
- IsUpper(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsLowSurrogate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the specified Char object is a low surrogate.

Overloads

Expand table

IsLowSurrogate(Char)	Indicates whether the specified Char object is a low surrogate.
IsLowSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a low surrogate.

Examples

The following code example demonstrates the IsHighSurrogate, IsLowSurrogate, and IsSurrogatePair methods.

C#

```
// This example demonstrates the Char.IsLowSurrogate() method
//                                     IsHighSurrogate() method
//                                     IsSurrogatePair() method
using System;

class Sample
{
    public static void Main()
    {
    char cHigh = '\uD800';
    char cLow  = '\uDC00';
    string s1  = new String(new char[] {'a', '\uD800', '\uDC00', 'z'});
    string divider = String.Concat( Environment.NewLine, new String('-',
70),
                                    Environment.NewLine);

    Console.WriteLine();
    Console.WriteLine("Hexadecimal code point of the character, cHigh:
```

```
{0:X4}", (int)cHigh);
    Console.WriteLine("Hexadecimal code point of the character, cLow:
{0:X4}", (int)cLow);
    Console.WriteLine();
    Console.WriteLine("Characters in string, s1: 'a', high surrogate, low
surrogate, 'z'");
    Console.WriteLine("Hexadecimal code points of the characters in string,
s1: ");
    for(int i = 0; i < s1.Length; i++)
        {
        Console.WriteLine("s1[{0}] = {1:X4} ", i, (int)s1[i]);
        }
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a high
surrogate?");
    Console.WriteLine("A1) cLow?  - {0}", Char.IsHighSurrogate(cLow));
    Console.WriteLine("A2) cHigh? - {0}", Char.IsHighSurrogate(cHigh));
    Console.WriteLine("A3) s1[0]? - {0}", Char.IsHighSurrogate(s1, 0));
    Console.WriteLine("A4) s1[1]? - {0}", Char.IsHighSurrogate(s1, 1));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a low
surrogate?");
    Console.WriteLine("B1) cLow?  - {0}", Char.IsLowSurrogate(cLow));
    Console.WriteLine("B2) cHigh? - {0}", Char.IsLowSurrogate(cHigh));
    Console.WriteLine("B3) s1[0]? - {0}", Char.IsLowSurrogate(s1, 0));
    Console.WriteLine("B4) s1[2]? - {0}", Char.IsLowSurrogate(s1, 2));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following pairs of characters a
surrogate pair?");
    Console.WriteLine("C1) cHigh and cLow?  - {0}",
Char.IsSurrogatePair(cHigh, cLow));
    Console.WriteLine("C2) s1[0] and s1[1]? - {0}", Char.IsSurrogatePair(s1,
0));
    Console.WriteLine("C3) s1[1] and s1[2]? - {0}", Char.IsSurrogatePair(s1,
1));
    Console.WriteLine("C4) s1[2] and s1[3]? - {0}", Char.IsSurrogatePair(s1,
2));
    Console.WriteLine(divider);
    }
}
/*
This example produces the following results:

Hexadecimal code point of the character, cHigh: D800
Hexadecimal code point of the character, cLow:  DC00

Characters in string, s1: 'a', high surrogate, low surrogate, 'z'
Hexadecimal code points of the characters in string, s1:
s1[0] = 0061
s1[1] = D800
s1[2] = DC00
s1[3] = 007A
```

```
-----------------------------------------------------------------------

Is each of the following characters a high surrogate?
A1) cLow?  - False
A2) cHigh? - True
A3) s1[0]? - False
A4) s1[1]? - True

-----------------------------------------------------------------------

Is each of the following characters a low surrogate?
B1) cLow?  - True
B2) cHigh? - False
B3) s1[0]? - False
B4) s1[2]? - True

-----------------------------------------------------------------------

Is each of the following pairs of characters a surrogate pair?
C1) cHigh and cLow?  - True
C2) s1[0] and s1[1]? - False
C3) s1[1] and s1[2]? - True
C4) s1[2] and s1[3]? - False

-----------------------------------------------------------------------

*/
```

IsLowSurrogate(Char)

Indicates whether the specified Char object is a low surrogate.

C#

```
public static bool IsLowSurrogate (char c);
```

Parameters

c Char

The character to evaluate.

Returns

Boolean

`true` if the numeric value of the `c` parameter ranges from U+DC00 through U+DFFF; otherwise, `false`.

Remarks

In addition to representing single characters using a 16-bit code point, UTF-16 encoding allows abstract characters to be represented using two 16-bit code points, which is known as a surrogate pair. The second element in this pair is the low surrogate. Its code point can range from U+DC00 to U+DFFF. An individual surrogate has no interpretation of its own; it is meaningful only when used as part of a surrogate pair.

See also

- Rune

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsLowSurrogate(String, Int32)

Indicates whether the Char object at the specified position in a string is a low surrogate.

C#

```
public static bool IsLowSurrogate (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the numeric value of the specified character in the `s` parameter ranges from U+DC00 through U+DFFF; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is not a position within `s`.

Remarks

The `index` parameter is zero-based.

In addition to representing single characters using a 16-bit code point, UTF-16 encoding allows abstract characters to be represented using two 16-bit code points, which is known as a surrogate pair. The second element in this pair is the low surrogate. Its code point can range from U+DC00 to U+DFFF. An individual surrogate has no interpretation of its own; it is meaningful only when used as part of a surrogate pair.

See also

- Rune

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a number.

Overloads

Expand table

IsNumber(Char)	Indicates whether the specified Unicode character is categorized as a number.
IsNumber(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a number.

IsNumber(Char)

Indicates whether the specified Unicode character is categorized as a number.

C#

```
public static bool IsNumber (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a number; otherwise, `false`.

Examples

The following example demonstrates IsNumber.

```C#
using System;

public class IsNumberSample {
    public static void Main() {
        string str = "non-numeric";

        Console.WriteLine(Char.IsNumber('8'));      // Output: "True"
        Console.WriteLine(Char.IsNumber(str, 3));   // Output: "False"
    }
}
```

Remarks

This method determines whether a Char is of any numeric Unicode category. In addition to including digits, numbers include characters, fractions, subscripts, superscripts, Roman numerals, currency numerators, and encircled numbers. This method contrasts with the IsDigit method, which determines whether a Char is a radix-10 digit.

> **Important**
>
> The **IsNumber(Char)** method is not intended to determine whether a string consists of numeric characters (for example, by calling the method for each character in a string). To determine whether a string consists of numeric characters, call one of the overloads of the `TryParse` method (such as **Int32.TryParse** or **Double.TryParse** of an integral or floating point type.

Valid numbers are members of the UnicodeCategory.DecimalDigitNumber, UnicodeCategory.LetterNumber, or UnicodeCategory.OtherNumber category.

The IsNumber(Char) method assumes that `c` corresponds to a single linguistic character and checks whether that character represents a number. However, some numbers in the Unicode standard are represented by two Char objects that form a surrogate pair. For example, the Aegean numbering system consists of code points U+10107 through U+10133. The following example uses the ConvertFromUtf32 method to instantiate a string that represents AEGEAN NUMBER ONE. As the output

from the example shows, the IsNumber(Char) method returns `false` if it is passed either a high surrogate or a low surrogate of this character.

C#

```
int utf32 = 0x10107;      // AEGEAN NUMBER ONE
string surrogate = Char.ConvertFromUtf32(utf32);
foreach (var ch in surrogate)
   Console.WriteLine("U+{0:X4}: {1}", Convert.ToUInt16(ch),
                                      Char.IsNumber(ch));

// The example displays the following output:
//       U+D800: False
//       U+DD07: False
```

See also

- IsNumber(Rune)
- Boolean
- IsDigit(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsNumber(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a number.

C#

```
public static bool IsNumber (string s, int index);
```

Parameters

s String

A string.

index Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a number; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Examples

The following example demonstrates IsNumber.

C#

```
using System;

public class IsNumberSample {
    public static void Main() {
        string str = "non-numeric";

        Console.WriteLine(Char.IsNumber('8'));      // Output: "True"
        Console.WriteLine(Char.IsNumber(str, 3));   // Output: "False"
    }
}
```

Remarks

This method determines whether a Char is of any numeric Unicode category. In addition to including digits, numbers include characters, fractions, subscripts, superscripts, Roman numerals, currency numerators, and encircled numbers. This method contrasts with the IsDigit method, which determines whether a Char is a radix-10 digit.

Character positions in a string are indexed starting from zero.

> ⓘ **Important**
>
> The **IsNumber(String, Int32)** method is not intended to determine whether a string consists of numeric characters (for example, by calling the method for each character in a string). To determine whether a string consists of numeric characters, call one of the overloads of the `TryParse` method (such as **Int32.TryParse** or **Double.TryParse** of an integral or floating point type.

Valid numbers are members of the UnicodeCategory.DecimalDigitNumber, UnicodeCategory.LetterNumber, or UnicodeCategory.OtherNumber category.

If the Char object at position `index` is the first character of a valid surrogate pair, the IsNumber(String, Int32) method determines whether the surrogate pair forms a numeric digit. For example, the Aegean numbering system consists of code points U+10107 through U+10133. The following example uses the ConvertFromUtf32 method to instantiate a string that represents AEGEAN NUMBER ONE. As the output from the example shows, the IsNumber(String, Int32) method returns `true` if it is passed the high surrogate of AEGEAN NUMBER ONE. However, if it is passed the low surrogate, it considers only the category of the low surrogate and returns `false`.

C#

```
int utf32 = 0x10107;      // AEGEAN NUMBER ONE
string surrogate = Char.ConvertFromUtf32(utf32);
for (int ctr = 0; ctr < surrogate.Length; ctr++)
   Console.WriteLine("U+{0:X4} at position {1}: {2}",
                     Convert.ToUInt16(surrogate[ctr]), ctr,
                     Char.IsNumber(surrogate, ctr));
// The example displays the following output:
//       U+D800 at position 0: True
//       U+DD07 at position 1: False
```

See also

- Boolean

- String
- IsDigit(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsPunctuation Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a punctuation mark.

Overloads

Expand table

IsPunctuation(Char)	Indicates whether the specified Unicode character is categorized as a punctuation mark.
IsPunctuation(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a punctuation mark.

Examples

The following code example demonstrates IsPunctuation.

C#

```
using System;

public class IsPunctuationSample {
    public static void Main() {
        char ch = '.';

        Console.WriteLine(Char.IsPunctuation(ch));                    //
Output: "True"
        Console.WriteLine(Char.IsPunctuation("no punctuation", 3));     //
Output: "False"
    }
}
```

Remarks

Valid punctuation marks are members of the following categories in UnicodeCategory: `ConnectorPunctuation`, `DashPunctuation`, `OpenPunctuation`, `ClosePunctuation`, `InitialQuotePunctuation`, `FinalQuotePunctuation`, or `OtherPunctuation`. They include characters with the Unicode code points listed in the following table.

Expand table

U+0021 to U+0023	U+060C and U+060D	U+1800 to U+180A	U+3014 to U+301F
U+0025 to U+002A	U+061B	U+1944 and U+1945	U+3030
U+002C to U+002F	U+061E and U+061F	U+19DE and U+19DF	U+303D
U+003A and U+003B	U+066A to U+066D	U+1A1E and U+1A1F	U+30A0
U+003F and U+0040	U+06D4	U+1B5A to U+1B60	U+30FB
U+005B to U+005D	U+0700 to U+070D	U+2010 to U+2027	U+A874 to U+A877
U+005F	U+07F7 to U+07F9	U+2030 to U+2043	U+FD3E and U+FD3F
U+007B	U+0964 and U+0965	U+2045 to U+2051	U+FE10 to U+FE19
U+007D	U+0970	U+2053 to U+205E	U+FE30 to U+FE52
U+00A1	U+0DF4	U+207D and U+207E	U+FE54 to U+FE61
U+00AB	U+0E4F to U+0E5B	U+208D and U+208E	U+FE63
U+00AD	U+0F04 to U+0F12	U+2329 and U+232A	U+FE68
U+00B7	U+0F3A to U+0F3D	U+2768 to U+2775	U+FE6A and U+FE6B
U+00BB	U+0F85	U+27C5 to U+27C6	U+FF01 to U+FF03
U+00BF	U+0FD0 and U+0FD1	U+27E6 to U+27EB	U+FF05 to U+FF0A
U+037E	U+104A to U+104F	U+2983 to U+2998	U+FF0C to U+FF0F
U+0387	U+10FB	U+29D8 to U+29DB	U+FF1A and U+FF1B
U+055A to U+055F	U+1361 to U+1368	U+29FC and U+29FD	U+FF1F and U+FF20
U+0589 and U+058A	U+166D and U+166E	U+2CF9 to U+2CFC	U+FF3B to U+FF3D
U+05BE	U+169B and U+169C	U+2CFE and U+2CFF	U+FF3F
U+05C0	U+16EB to U+16ED	U+2E00 to U+2E17	U+FF5B
U+05C3	U+1735 and U+1736	U+2E1C and U+2E1D	U+FF5D

U+05C6	U+17D4 to U+17D6	U+3001 to U+3003	U+FF5F to U+FF65
U+05F3 and U+05F4	U+17D8 to U+17DA	U+3008 to U+3011	

IsPunctuation(Char)

Indicates whether the specified Unicode character is categorized as a punctuation mark.

```
C#
public static bool IsPunctuation (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a punctuation mark; otherwise, `false`.

Remarks

Valid punctuation marks are members of the following categories in UnicodeCategory: `ConnectorPunctuation`, `DashPunctuation`, `OpenPunctuation`, `ClosePunctuation`, `InitialQuotePunctuation`, `FinalQuotePunctuation`, or `OtherPunctuation`. They include characters with the Unicode code points listed in the following table.

Expand table

U+0021 to U+0023	U+060C and U+060D	U+1800 to U+180A	U+3014 to U+301F
U+0025 to U+002A	U+061B	U+1944 and U+1945	U+3030
U+002C to U+002F	U+061E and U+061F	U+19DE and U+19DF	U+303D
U+003A and U+003B	U+066A to U+066D	U+1A1E and U+1A1F	U+30A0

U+003F and U+0040	U+06D4	U+1B5A to U+1B60	U+30FB
U+005B to U+005D	U+0700 to U+070D	U+2010 to U+2027	U+A874 to U+A877
U+005F	U+07F7 to U+07F9	U+2030 to U+2043	U+FD3E and U+FD3F
U+007B	U+0964 and U+0965	U+2045 to U+2051	U+FE10 to U+FE19
U+007D	U+0970	U+2053 to U+205E	U+FE30 to U+FE52
U+00A1	U+0DF4	U+207D and U+207E	U+FE54 to U+FE61
U+00AB	U+0E4F to U+0E5B	U+208D and U+208E	U+FE63
U+00AD	U+0F04 to U+0F12	U+2329 and U+232A	U+FE68
U+00B7	U+0F3A to U+0F3D	U+2768 to U+2775	U+FE6A and U+FE6B
U+00BB	U+0F85	U+27C5 to U+27C6	U+FF01 to U+FF03
U+00BF	U+0FD0 and U+0FD1	U+27E6 to U+27EB	U+FF05 to U+FF0A
U+037E	U+104A to U+104F	U+2983 to U+2998	U+FF0C to U+FF0F
U+0387	U+10FB	U+29D8 to U+29DB	U+FF1A and U+FF1B
U+055A to U+055F	U+1361 to U+1368	U+29FC and U+29FD	U+FF1F and U+FF20
U+0589 and U+058A	U+166D and U+166E	U+2CF9 to U+2CFC	U+FF3B to U+FF3D
U+05BE	U+169B and U+169C	U+2CFE and U+2CFF	U+FF3F
U+05C0	U+16EB to U+16ED	U+2E00 to U+2E17	U+FF5B
U+05C3	U+1735 and U+1736	U+2E1C and U+2E1D	U+FF5D
U+05C6	U+17D4 to U+17D6	U+3001 to U+3003	U+FF5F to U+FF65
U+05F3 and U+05F4	U+17D8 to U+17DA	U+3008 to U+3011	

See also

- IsPunctuation(Rune)
- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsPunctuation(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a punctuation mark.

C#

```
public static bool IsPunctuation (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a punctuation mark; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

Valid punctuation marks are members of the following categories in UnicodeCategory: `ConnectorPunctuation`, `DashPunctuation`, `OpenPunctuation`, `ClosePunctuation`, `InitialQuotePunctuation`, `FinalQuotePunctuation`, or `OtherPunctuation`. They include characters with the Unicode code points listed in the following table.

Expand table

U+0021 to U+0023	U+060C and U+060D	U+1800 to U+180A	U+3014 to U+301F
U+0025 to U+002A	U+061B	U+1944 and U+1945	U+3030
U+002C to U+002F	U+061E and U+061F	U+19DE and U+19DF	U+303D
U+003A and U+003B	U+066A to U+066D	U+1A1E and U+1A1F	U+30A0
U+003F and U+0040	U+06D4	U+1B5A to U+1B60	U+30FB
U+005B to U+005D	U+0700 to U+070D	U+2010 to U+2027	U+A874 to U+A877
U+005F	U+07F7 to U+07F9	U+2030 to U+2043	U+FD3E and U+FD3F
U+007B	U+0964 and U+0965	U+2045 to U+2051	U+FE10 to U+FE19
U+007D	U+0970	U+2053 to U+205E	U+FE30 to U+FE52
U+00A1	U+0DF4	U+207D and U+207E	U+FE54 to U+FE61
U+00AB	U+0E4F to U+0E5B	U+208D and U+208E	U+FE63
U+00AD	U+0F04 to U+0F12	U+2329 and U+232A	U+FE68
U+00B7	U+0F3A to U+0F3D	U+2768 to U+2775	U+FE6A and U+FE6B
U+00BB	U+0F85	U+27C5 to U+27C6	U+FF01 to U+FF03
U+00BF	U+0FD0 and U+0FD1	U+27E6 to U+27EB	U+FF05 to U+FF0A
U+037E	U+104A to U+104F	U+2983 to U+2998	U+FF0C to U+FF0F
U+0387	U+10FB	U+29D8 to U+29DB	U+FF1A and U+FF1B
U+055A to U+055F	U+1361 to U+1368	U+29FC and U+29FD	U+FF1F and U+FF20
U+0589 and U+058A	U+166D and U+166E	U+2CF9 to U+2CFC	U+FF3B to U+FF3D

U+05BE	U+169B and U+169C	U+2CFE and U+2CFF	U+FF3F
U+05C0	U+16EB to U+16ED	U+2E00 to U+2E17	U+FF5B
U+05C3	U+1735 and U+1736	U+2E1C and U+2E1D	U+FF5D
U+05C6	U+17D4 to U+17D6	U+3001 to U+3003	U+FF5F to U+FF65
U+05F3 and U+05F4	U+17D8 to U+17DA	U+3008 to U+3011	

See also

- Boolean
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsSeparator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a separator character.

Overloads

Expand table

IsSeparator(Char)	Indicates whether the specified Unicode character is categorized as a separator character.
IsSeparator(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a separator character.

Remarks

The Unicode standard recognizes three subcategories of separators:

- Space separators (the UnicodeCategory.SpaceSeparator category), which includes characters such as \u0020.
- Line separators (the UnicodeCategory.LineSeparator category), which includes \u2028.
- Paragraph separators (the UnicodeCategory.ParagraphSeparator category), which includes \u2029.

> **Note**
>
> The Unicode standard classifies the characters \u000A (LF), \u000C (FF), and \u000D (CR) as control characters (members of the **UnicodeCategory.Control** category), not as separator characters.

IsSeparator(Char)

Indicates whether the specified Unicode character is categorized as a separator character.

```
C#

public static bool IsSeparator (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a separator character; otherwise, `false`.

Examples

The following example lists the Char objects that are classified as separator characters.

```
C#

using System;

public class Class1
{
   public static void Main()
   {
      for (int ctr = (int)(Char.MinValue); ctr <= (int)(Char.MaxValue);
ctr++)
      {
         char ch = (Char)ctr;
         if (Char.IsSeparator(ch))
            Console.WriteLine(@"\u{(int)ch:X4}
({Char.GetUnicodeCategory(ch)})");
      }
   }
}
// The example displays the following output:
//       \u0020 (SpaceSeparator)
//       \u00A0 (SpaceSeparator)
```

```
//       \u1680 (SpaceSeparator)
//       \u180E (SpaceSeparator)
//       \u2000 (SpaceSeparator)
//       \u2001 (SpaceSeparator)
//       \u2002 (SpaceSeparator)
//       \u2003 (SpaceSeparator)
//       \u2004 (SpaceSeparator)
//       \u2005 (SpaceSeparator)
//       \u2006 (SpaceSeparator)
//       \u2007 (SpaceSeparator)
//       \u2008 (SpaceSeparator)
//       \u2009 (SpaceSeparator)
//       \u200A (SpaceSeparator)
//       \u2028 (LineSeparator)
//       \u2029 (ParagraphSeparator)
//       \u202F (SpaceSeparator)
//       \u205F (SpaceSeparator)
//       \u3000 (SpaceSeparator)
```

Remarks

The Unicode standard recognizes three subcategories of separators:

- Space separators (the UnicodeCategory.SpaceSeparator category), which includes characters such as \u0020.
- Line separators (the UnicodeCategory.LineSeparator category), which includes \u2028.
- Paragraph separators (the UnicodeCategory.ParagraphSeparator category), which includes \u2029.

> **Note**
>
> The Unicode standard classifies the characters \u000A (LF), \u000C (FF), and \u000D (CR) as control characters (members of the **UnicodeCategory.Control** category), not as separator characters.

See also

- IsSeparator(Rune)
- UnicodeCategory

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsSeparator(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a separator character.

```
C#
public static bool IsSeparator (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a separator character; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Examples

The following example demonstrates IsSeparator.

C#

```
using System;

public class IsSeparatorSample {
    public static void Main() {
        string str = "twain1 twain2";

        Console.WriteLine(Char.IsSeparator('a'));       // Output:
"False"
        Console.WriteLine(Char.IsSeparator(str, 6));    // Output: "True"
    }
}
```

Remarks

Character positions in a string are indexed starting from zero.

The Unicode standard recognizes three subcategories of separators:

- Space separators (the UnicodeCategory.SpaceSeparator category), which includes characters such as \u0020.
- Line separators (the UnicodeCategory.LineSeparator category), which includes \u2028.
- Paragraph separators (the UnicodeCategory.ParagraphSeparator category), which includes \u2029.

> **Note**
>
> The Unicode standard classifies the characters \u000A (LF), \u000C (FF), and \u000D (CR) as control characters (members of the **UnicodeCategory.Control** category), not as separator characters.

See also

- Boolean
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsSurrogate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a character has a surrogate code unit.

Overloads

Expand table

IsSurrogate(String, Int32)	Indicates whether the character at the specified position in a specified string has a surrogate code unit.
IsSurrogate(Char)	Indicates whether the specified character has a surrogate code unit.

Examples

The following example demonstrates the IsSurrogate method.

C#

```
using System;

public class IsSurrogateSample {
    public static void Main() {
        string str = "\U00010F00"; // Unicode values between 0x10000 and
0x10FFF are represented by two 16-bit "surrogate" characters

        Console.WriteLine(Char.IsSurrogate('a'));       // Output: "False"
        Console.WriteLine(Char.IsSurrogate(str, 0));    // Output: "True"
    }
}
```

IsSurrogate(String, Int32)

Indicates whether the character at the specified position in a specified string has a surrogate code unit.

```C#
public static bool IsSurrogate (string s, int index);
```

Parameters

s String

A string.

index Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a either a high surrogate or a low surrogate; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

A surrogate is a Char object with a UTF-16 code unit in the range from U+D800 to U+DFFF. Each character with a code unit in this range belongs to the UnicodeCategory.Surrogate category. The individual surrogate code unit has no interpretation of its own, but has meaning only when used as part of a surrogate pair. For more information about surrogate pairs, see the Unicode Standard at the Unicode home page.

See also

- Rune

- IsHighSurrogate
- IsLowSurrogate
- IsSurrogatePair

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsSurrogate(Char)

Indicates whether the specified character has a surrogate code unit.

C#

```
public static bool IsSurrogate (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is either a high surrogate or a low surrogate; otherwise, `false`.

Remarks

A surrogate is a Char object with a UTF-16 code unit in the range from U+D800 to U+DFFF. Each character with a code unit in this range belongs to the

UnicodeCategory.Surrogate category. The individual surrogate code unit has no interpretation of its own, but has meaning only when used as part of a surrogate pair. For more information about surrogate pairs, see the Unicode Standard at the Unicode home page.

See also

- Rune
- IsHighSurrogate
- IsLowSurrogate
- IsSurrogatePair

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsSurrogatePair Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two specified Char objects form a surrogate pair.

Overloads

Expand table

IsSurrogatePair(Char, Char)	Indicates whether the two specified Char objects form a surrogate pair.
IsSurrogatePair(String, Int32)	Indicates whether two adjacent Char objects at a specified position in a string form a surrogate pair.

Examples

The following code example demonstrates the IsHighSurrogate, IsLowSurrogate, and IsSurrogatePair methods.

C#

```
// This example demonstrates the Char.IsLowSurrogate() method
//                                IsHighSurrogate() method
//                                IsSurrogatePair() method
using System;

class Sample
{
    public static void Main()
    {
    char cHigh = '\uD800';
    char cLow  = '\uDC00';
    string s1  = new String(new char[] {'a', '\uD800', '\uDC00', 'z'});
    string divider = String.Concat( Environment.NewLine, new String('-',
70),
                                    Environment.NewLine);

    Console.WriteLine();
```

```
    Console.WriteLine("Hexadecimal code point of the character, cHigh:
{0:X4}", (int)cHigh);
    Console.WriteLine("Hexadecimal code point of the character, cLow:
{0:X4}", (int)cLow);
    Console.WriteLine();
    Console.WriteLine("Characters in string, s1: 'a', high surrogate, low
surrogate, 'z'");
    Console.WriteLine("Hexadecimal code points of the characters in string,
s1: ");
    for(int i = 0; i < s1.Length; i++)
        {
        Console.WriteLine("s1[{0}] = {1:X4} ", i, (int)s1[i]);
        }
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a high
surrogate?");
    Console.WriteLine("A1) cLow?  - {0}", Char.IsHighSurrogate(cLow));
    Console.WriteLine("A2) cHigh? - {0}", Char.IsHighSurrogate(cHigh));
    Console.WriteLine("A3) s1[0]? - {0}", Char.IsHighSurrogate(s1, 0));
    Console.WriteLine("A4) s1[1]? - {0}", Char.IsHighSurrogate(s1, 1));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following characters a low
surrogate?");
    Console.WriteLine("B1) cLow?  - {0}", Char.IsLowSurrogate(cLow));
    Console.WriteLine("B2) cHigh? - {0}", Char.IsLowSurrogate(cHigh));
    Console.WriteLine("B3) s1[0]? - {0}", Char.IsLowSurrogate(s1, 0));
    Console.WriteLine("B4) s1[2]? - {0}", Char.IsLowSurrogate(s1, 2));
    Console.WriteLine(divider);

    Console.WriteLine("Is each of the following pairs of characters a
surrogate pair?");
    Console.WriteLine("C1) cHigh and cLow?  - {0}",
Char.IsSurrogatePair(cHigh, cLow));
    Console.WriteLine("C2) s1[0] and s1[1]? - {0}", Char.IsSurrogatePair(s1,
0));
    Console.WriteLine("C3) s1[1] and s1[2]? - {0}", Char.IsSurrogatePair(s1,
1));
    Console.WriteLine("C4) s1[2] and s1[3]? - {0}", Char.IsSurrogatePair(s1,
2));
    Console.WriteLine(divider);
    }
}
/*
This example produces the following results:

Hexadecimal code point of the character, cHigh: D800
Hexadecimal code point of the character, cLow:  DC00

Characters in string, s1: 'a', high surrogate, low surrogate, 'z'
Hexadecimal code points of the characters in string, s1:
s1[0] = 0061
s1[1] = D800
s1[2] = DC00
```

```
s1[3] = 007A

----------------------------------------------------------------------

Is each of the following characters a high surrogate?
A1) cLow?  - False
A2) cHigh? - True
A3) s1[0]? - False
A4) s1[1]? - True

----------------------------------------------------------------------

Is each of the following characters a low surrogate?
B1) cLow?  - True
B2) cHigh? - False
B3) s1[0]? - False
B4) s1[2]? - True

----------------------------------------------------------------------

Is each of the following pairs of characters a surrogate pair?
C1) cHigh and cLow?  - True
C2) s1[0] and s1[1]? - False
C3) s1[1] and s1[2]? - True
C4) s1[2] and s1[3]? - False

----------------------------------------------------------------------

*/
```

IsSurrogatePair(Char, Char)

Indicates whether the two specified Char objects form a surrogate pair.

```
C#

public static bool IsSurrogatePair (char highSurrogate, char
lowSurrogate);
```

Parameters

highSurrogate Char

The character to evaluate as the high surrogate of a surrogate pair.

lowSurrogate Char

The character to evaluate as the low surrogate of a surrogate pair.

Returns

Boolean

`true` if the numeric value of the `highSurrogate` parameter ranges from U+D800 through U+DBFF, and the numeric value of the `lowSurrogate` parameter ranges from U+DC00 through U+DFFF; otherwise, `false`.

Remarks

Ordinarily, a single character is represented by a single 16-bit Unicode code unit. UTF-16 encoding also supports surrogate pairs, which allow a single abstract character to be represented by two 16-bit code units. The first code unit, whose value can range from U+D800 to U+DBFF, is the high surrogate. The second code unit, whose value can range from U+DC00 to U+DFFF, is the low surrogate. Individual surrogate code points have no interpretation of their own. For more information about surrogates and the Unicode Standard, see the Unicode home page.

See also

- Rune
- IsHighSurrogate(Char)
- IsLowSurrogate(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsSurrogatePair(String, Int32)

Indicates whether two adjacent Char objects at a specified position in a string form a surrogate pair.

```
C#

public static bool IsSurrogatePair (string s, int index);
```

Parameters

s String

A string.

index Int32

The starting position of the pair of characters to evaluate within `s`.

Returns

Boolean

`true` if the `s` parameter includes adjacent characters at positions `index` and `index` + 1, and the numeric value of the character at position `index` ranges from U+D800 through U+DBFF, and the numeric value of the character at position `index`+1 ranges from U+DC00 through U+DFFF; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is not a position within `s`.

Remarks

Ordinarily, a single character is represented by a single 16-bit Unicode code unit. UTF-16 encoding also supports surrogate pairs, which allow a single abstract character to be represented by two 16-bit code units. The first code unit, whose value can range from U+D800 to U+DBFF, is the high surrogate. The second code unit, whose value can range from U+DC00 to U+DFFF, is the low surrogate. Individual surrogate code points have no interpretation of their own. For more

information about surrogates and the Unicode Standard, see the Unicode home page.

See also

- Rune
- IsHighSurrogate(Char)
- IsLowSurrogate(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IsSymbol Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as a symbol character.

Overloads

Expand table

IsSymbol(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a symbol character.
IsSymbol(Char)	Indicates whether the specified Unicode character is categorized as a symbol character.

Examples

The following code example demonstrates IsSymbol.

C#

```
using System;

public class IsSymbolSample {
    public static void Main() {
        string str = "non-symbolic characters";

        Console.WriteLine(Char.IsSymbol('+'));      // Output: "True"
        Console.WriteLine(Char.IsSymbol(str, 8));   // Output: "False"
    }
}
```

Remarks

Valid symbols are members of the following categories in UnicodeCategory: `MathSymbol`, `CurrencySymbol`, `ModifierSymbol`, and `OtherSymbol`.

Symbols in the Unicode standard are a loosely defined set of characters that include the following:

- Currency symbols.
- Letterlike symbols, which include a set of mathematical alphanumeric symbols as well as symbols such as ℅, №, and ™.
- Number forms, such as subscripts and superscripts.
- Mathematical operators and arrows.
- Geometric symbols.
- Technical symbols.
- Braille patterns.
- Dingbats.

IsSymbol(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as a symbol character.

```
C#
public static bool IsSymbol (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is a symbol character; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

Valid symbols are members of the following categories in System.Globalization.UnicodeCategory: `MathSymbol`, `CurrencySymbol`, `ModifierSymbol`, and `OtherSymbol`.

Symbols in the Unicode standard are a loosely defined set of characters that include the following:

- Currency symbols.
- Letterlike symbols, which include a set of mathematical alphanumeric symbols as well as symbols such as ℅, №, and ™.
- Number forms, such as subscripts and superscripts.
- Mathematical operators and arrows.
- Geometric symbols.
- Technical symbols.
- Braille patterns.
- Dingbats.

See also

- Boolean
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsSymbol(Char)

Indicates whether the specified Unicode character is categorized as a symbol character.

```
C#

public static bool IsSymbol (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is a symbol character; otherwise, `false`.

Remarks

Valid symbols are members of the following categories in UnicodeCategory: `MathSymbol`, `CurrencySymbol`, `ModifierSymbol`, and `OtherSymbol`.

Symbols in the Unicode standard are a loosely defined set of characters that include the following:

- Currency symbols.

- Letterlike symbols, which include a set of mathematical alphanumeric symbols as well as symbols such as ℅, №, and ™.
- Number forms, such as subscripts and superscripts.
- Mathematical operators and arrows.
- Geometric symbols.
- Technical symbols.
- Braille patterns.
- Dingbats.

See also

- IsSymbol(Rune)
- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Char.IsUpper Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as an uppercase letter.

Overloads

Expand table

IsUpper(Char)	Indicates whether the specified Unicode character is categorized as an uppercase letter.
IsUpper(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as an uppercase letter.

IsUpper(Char)

Indicates whether the specified Unicode character is categorized as an uppercase letter.

C#

```
public static bool IsUpper (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is an uppercase letter; otherwise, `false`.

Remarks

Valid uppercase letters are members of the following category in UnicodeCategory: `UppercaseLetter`.

See also

- IsUpper(Rune)
- IsLower(Rune)
- Boolean
- IsLower(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsUpper(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as an uppercase letter.

C#

```
public static bool IsUpper (string s, int index);
```

Parameters

s String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is an uppercase letter; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

Valid uppercase letters are members of the following category in UnicodeCategory: `UppercaseLetter`.

See also

- Boolean
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IsWhiteSpace Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a Unicode character is categorized as white space.

Overloads

Expand table

IsWhiteSpace(Char)	Indicates whether the specified Unicode character is categorized as white space.
IsWhiteSpace(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as white space.

Examples

The following example demonstrates the IsWhiteSpace(Char) method.

C#

```
using System;

public class IsWhiteSpaceSample {
    public static void Main() {
        string str = "black matter";

        Console.WriteLine(Char.IsWhiteSpace('A'));      // Output: "False"
        Console.WriteLine(Char.IsWhiteSpace(str, 5));   // Output: "True"
    }
}
```

Remarks

White space characters are the following Unicode characters:

- Members of the UnicodeCategory.SpaceSeparator category, which includes the characters SPACE (U+0020), NO-BREAK SPACE (U+00A0), OGHAM SPACE MARK (U+1680), EN QUAD (U+2000), EM QUAD (U+2001), EN SPACE (U+2002), EM SPACE (U+2003), THREE-PER-EM SPACE (U+2004), FOUR-PER-EM SPACE (U+2005), SIX-PER-EM SPACE (U+2006), FIGURE SPACE (U+2007), PUNCTUATION SPACE (U+2008), THIN SPACE (U+2009), HAIR SPACE (U+200A), NARROW NO-BREAK SPACE (U+202F), MEDIUM MATHEMATICAL SPACE (U+205F), and IDEOGRAPHIC SPACE (U+3000).
- Members of the UnicodeCategory.LineSeparator category, which consists solely of the LINE SEPARATOR character (U+2028).
- Members of the UnicodeCategory.ParagraphSeparator category, which consists solely of the PARAGRAPH SEPARATOR character (U+2029).
- The characters CHARACTER TABULATION (U+0009), LINE FEED (U+000A), LINE TABULATION (U+000B), FORM FEED (U+000C), CARRIAGE RETURN (U+000D), and NEXT LINE (U+0085).

IsWhiteSpace(Char)

Indicates whether the specified Unicode character is categorized as white space.

```
C#
public static bool IsWhiteSpace (char c);
```

Parameters

c Char

The Unicode character to evaluate.

Returns

Boolean

`true` if `c` is white space; otherwise, `false`.

Remarks

White space characters are the following Unicode characters:

- Members of the UnicodeCategory.SpaceSeparator category, which includes the characters SPACE (U+0020), NO-BREAK SPACE (U+00A0), OGHAM SPACE MARK (U+1680), EN QUAD (U+2000), EM QUAD (U+2001), EN SPACE (U+2002), EM SPACE (U+2003), THREE-PER-EM SPACE (U+2004), FOUR-PER-EM SPACE (U+2005), SIX-PER-EM SPACE (U+2006), FIGURE SPACE (U+2007), PUNCTUATION SPACE (U+2008), THIN SPACE (U+2009), HAIR SPACE (U+200A), NARROW NO-BREAK SPACE (U+202F), MEDIUM MATHEMATICAL SPACE (U+205F), and IDEOGRAPHIC SPACE (U+3000).
- Members of the UnicodeCategory.LineSeparator category, which consists solely of the LINE SEPARATOR character (U+2028).
- Members of the UnicodeCategory.ParagraphSeparator category, which consists solely of the PARAGRAPH SEPARATOR character (U+2029).
- The characters CHARACTER TABULATION (U+0009), LINE FEED (U+000A), LINE TABULATION (U+000B), FORM FEED (U+000C), CARRIAGE RETURN (U+000D), and NEXT LINE (U+0085).

See also

- IsWhiteSpace(Rune)
- Boolean
- UnicodeCategory

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsWhiteSpace(String, Int32)

Indicates whether the character at the specified position in a specified string is categorized as white space.

```
C#
public static bool IsWhiteSpace (string s, int index);
```

Parameters

`s` String

A string.

`index` Int32

The position of the character to evaluate in `s`.

Returns

Boolean

`true` if the character at position `index` in `s` is white space; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentOutOfRangeException

`index` is less than zero or greater than the last position in `s`.

Remarks

Character positions in a string are indexed starting from zero.

White space characters are the following Unicode characters:

- Members of the UnicodeCategory.SpaceSeparator category, which includes the characters SPACE (U+0020), NO-BREAK SPACE (U+00A0), OGHAM SPACE MARK (U+1680), EN QUAD (U+2000), EM QUAD (U+2001), EN SPACE (U+2002), EM SPACE (U+2003), THREE-PER-EM SPACE (U+2004), FOUR-PER-EM SPACE (U+2005), SIX-PER-EM SPACE (U+2006), FIGURE SPACE (U+2007), PUNCTUATION SPACE (U+2008), THIN SPACE (U+2009), HAIR SPACE (U+200A),

NARROW NO-BREAK SPACE (U+202F), MEDIUM MATHEMATICAL SPACE (U+205F), and IDEOGRAPHIC SPACE (U+3000).

- Members of the UnicodeCategory.LineSeparator category, which consists solely of the LINE SEPARATOR character (U+2028).
- Members of the UnicodeCategory.ParagraphSeparator category, which consists solely of the PARAGRAPH SEPARATOR character (U+2029).
- The characters CHARACTER TABULATION (U+0009), LINE FEED (U+000A), LINE TABULATION (U+000B), FORM FEED (U+000C), CARRIAGE RETURN (U+000D), and NEXT LINE (U+0085).

See also

- Boolean
- String
- UnicodeCategory

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Char.Parse(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified string to its equivalent Unicode character.

C#

```
public static char Parse (string s);
```

Parameters

s String

A string that contains a single character, or `null`.

Returns

Char

A Unicode character equivalent to the sole character in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

The length of `s` is not 1.

Examples

The following code example demonstrates Parse.

C#

```
using System;

public class ParseSample {
    public static void Main() {
        Console.WriteLine(Char.Parse("A")); // Output: 'A'
    }
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToString()
- Parsing Other Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ToLower Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a Unicode character to its lowercase equivalent.

Overloads

Expand table

ToLower(Char, CultureInfo)	Converts the value of a specified Unicode character to its lowercase equivalent using specified culture-specific formatting information.
ToLower(Char)	Converts the value of a Unicode character to its lowercase equivalent.

Examples

The following example demonstrates ToLower.

C#

```
using System;
using System.Globalization;

public class ToLowerSample {
    public static void Main() {
        Console.WriteLine(Char.ToLower('A'));              // Output: "a"
    }
}
```

ToLower(Char, CultureInfo)

Converts the value of a specified Unicode character to its lowercase equivalent using specified culture-specific formatting information.

C#

```
public static char ToLower (char c, System.Globalization.CultureInfo
culture);
```

Parameters

`c` Char

The Unicode character to convert.

`culture` CultureInfo

An object that supplies culture-specific casing rules.

Returns

Char

The lowercase equivalent of `c`, modified according to `culture`, or the unchanged value of `c`, if `c` is already lowercase or not alphabetic.

Exceptions

ArgumentNullException

`culture` is `null`.

Remarks

Use String.ToLower to convert a string to lowercase.

See also

- ToLower(Rune, CultureInfo)
- ToUpper(Rune, CultureInfo)
- CurrentCulture
- ToLower()
- ToUpper(Char, CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToLower(Char)

Converts the value of a Unicode character to its lowercase equivalent.

```
C#
public static char ToLower (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

Char

The lowercase equivalent of `c`, or the unchanged value of `c`, if `c` is already lowercase or not alphabetic.

Remarks

Casing rules are obtained from the current culture.

Use String.ToLower to convert a string to lowercase.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling character-casing and string-casing methods that substitute default values. Instead, you should call methods that require parameters to be explicitly specified. To convert a character to lowercase by using the casing conventions of the current

culture, call the ToLower(Char, CultureInfo) method overload with a value of CurrentCulture for its `culture` parameter.

See also

- ToLower(Rune, CultureInfo)
- ToUpper(Rune, CultureInfo)
- ToLower()
- ToUpper(Char, CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ToLowerInvariant(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a Unicode character to its lowercase equivalent using the casing rules of the invariant culture.

```
C#
public static char ToLowerInvariant (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

Char

The lowercase equivalent of the `c` parameter, or the unchanged value of `c`, if `c` is already lowercase or not alphabetic.

Remarks

If your application is unaffected by the current culture and depends on the case of a character changing in a predictable way, then use the ToLowerInvariant method. The ToLowerInvariant method is equivalent to `ToLower(Char, CultureInfo.InvariantCulture)`.

Use String.ToLowerInvariant to convert a string to lowercase.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToLowerInvariant(Rune)
- ToLower()
- ToUpper(Char, CultureInfo)
- InvariantCulture

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(Char)	Converts the specified Unicode character to its equivalent string representation.
ToString()	Converts the value of this instance to its equivalent string representation.

Examples

The following code example demonstrates ToString.

C#

```
using System;

public class ToStringSample {
    public static void Main() {
        char ch = 'a';
        Console.WriteLine(ch.ToString());       // Output: "a"

        Console.WriteLine(Char.ToString('b'));  // Output: "b"
    }
}
```

ToString(IFormatProvider)

Converts the value of this instance to its equivalent string representation using the specified culture-specific format information.

```
C#

public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

(Reserved) An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Remarks

The `provider` parameter is ignored; it does not participate in this operation.

See also

- Parse(String)
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToString(Char)

Converts the specified Unicode character to its equivalent string representation.

C#

```
public static string ToString (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

String

The string representation of the value of `c`.

See also

- Parse(String)
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the value of this instance to its equivalent string representation.

```
C#
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

See also

- Parse(String)
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ToUpper Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a Unicode character to its uppercase equivalent.

Overloads

Expand table

ToUpper(Char, CultureInfo)	Converts the value of a specified Unicode character to its uppercase equivalent using specified culture-specific formatting information.
ToUpper(Char)	Converts the value of a Unicode character to its uppercase equivalent.

ToUpper(Char, CultureInfo)

Converts the value of a specified Unicode character to its uppercase equivalent using specified culture-specific formatting information.

C#

```
public static char ToUpper (char c, System.Globalization.CultureInfo
culture);
```

Parameters

`c` Char

The Unicode character to convert.

`culture` CultureInfo

An object that supplies culture-specific casing rules.

Returns

Char

The uppercase equivalent of `c`, modified according to `culture`, or the unchanged value of `c` if `c` is already uppercase, has no uppercase equivalent, or is not alphabetic.

Exceptions

ArgumentNullException

`culture` is `null`.

Examples

The following example converts each character in an array to its uppercase equivalent for the en-US culture, the invariant culture, and the tr-TR culture. In this example, the uppercase equivalent of each lowercase letter is identical for all cultures except for one case. The lowercase "i" character (U+0069) converts to "I" (U+0049) in the en-US and invariant cultures, but to "İ" (U+0130) in the tr-TR culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo[] cultures= { CultureInfo.CreateSpecificCulture("en-
US"),
                                CultureInfo.InvariantCulture,
                                CultureInfo.CreateSpecificCulture("tr-
TR") };
      Char[] chars = {'ä', 'e', 'E', 'i', 'I' };

      Console.WriteLine("Character     en-US     Invariant     tr-TR");
      foreach (var ch in chars) {
         Console.Write("    {0}", ch);
         foreach (var culture in cultures)
            Console.Write("{0,12}", Char.ToUpper(ch, culture));

         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       Character     en-US     Invariant     tr-TR
//           ä           Ä           Ä           Ä
```

```
//          e           E           E           E
//          E           E           E           E
//          i           I           I           İ
//          I           I           I           I
```

Remarks

Use String.ToUpper to convert a string to uppercase.

See also

- ToUpper(Rune, CultureInfo)
- ToLower(Rune, CultureInfo)
- CurrentCulture
- ToUpper()
- ToLower(Char, CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToUpper(Char)

Converts the value of a Unicode character to its uppercase equivalent.

C#

```
public static char ToUpper (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

Char

The uppercase equivalent of `c`, or the unchanged value of `c` if `c` is already uppercase, has no uppercase equivalent, or is not alphabetic.

Examples

The following example converts each character in an array to its uppercase equivalent.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      char[] chars = { 'e', 'E', '6', ',', 'ж', 'ä' };
      foreach (var ch in chars)
          Console.WriteLine("{0} --> {1} {2}", ch, Char.ToUpper(ch),
                                ch == Char.ToUpper(ch) ? "(Same Character)" :
"" );
   }
}
// The example displays the following output:
//       e --> E
//       E --> E (Same Character)
//       6 --> 6 (Same Character)
//       , --> , (Same Character)
//       ж --> Ж
//       ä --> Ä
```

Remarks

Casing rules are obtained from the current culture.

Use String.ToUpper to convert a string to uppercase.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling character-casing and string-casing methods that substitute default values. Instead, you should call methods that require parameters to be explicitly specified. To convert a character to uppercase by using the casing conventions of the current

culture, call the ToUpper(Char, CultureInfo) method overload with a value of CurrentCulture for its `culture` parameter.

See also

- ToUpper(Rune, CultureInfo)
- ToLower(Rune, CultureInfo)
- ToUpper()
- ToLower(Char, CultureInfo)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ToUpperInvariant(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a Unicode character to its uppercase equivalent using the casing rules of the invariant culture.

```
C#
public static char ToUpperInvariant (char c);
```

Parameters

c Char

The Unicode character to convert.

Returns

Char

The uppercase equivalent of the `c` parameter, or the unchanged value of `c`, if `c` is already uppercase or not alphabetic.

Remarks

If your application is unaffected by the current culture and depends on the case of a character changing in a predictable way, use the ToUpperInvariant method. The ToUpperInvariant method is equivalent to `ToUpper(Char, CultureInfo.InvariantCulture)`.

Use String.ToUpperInvariant to convert a string to uppercase.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToUpperInvariant(Rune)
- ToUpper()
- ToUpper(Char, CultureInfo)
- InvariantCulture

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.TryParse(String, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified string to its equivalent Unicode character. A return code indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (string? s, out char result);
```

Parameters

s String

A string that contains a single character, or `null`.

result Char

When this method returns, contains a Unicode character equivalent to the sole character in `s`, if the conversion succeeded, or an undefined value if the conversion failed. The conversion fails if the `s` parameter is `null` or the length of `s` is not 1. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Examples

The following code example demonstrates overloads of the TryParse method for several base types, and the TryParseExact method for the DateTime base type.

```
C#
```

```
// This example demonstrates overloads of the TryParse method for
// several base types, and the TryParseExact method for DateTime.

// In most cases, this example uses the most complex overload; that is, the
overload
// with the most parameters for a particular type. If a complex overload
specifies
// null (Nothing in Visual Basic) for the IFormatProvider parameter,
formatting
// information is obtained from the culture associated with the current
thread.
// If a complex overload specifies the style parameter, the parameter value
is
// the default value used by the equivalent simple overload.

using System;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    bool     success;
    CultureInfo ci;
    string   nl = Environment.NewLine;
    string   msg1 =
             "This example demonstrates overloads of the TryParse method
for{0}" +
             "several base types, as well as the TryParseExact method for
DateTime.{0}";
    string   msg2 = "Non-numeric types:{0}";
    string   msg3 = "{0}Numeric types:{0}";
    string   msg4 = "{0}The following types are not CLS-compliant:{0}";

// Non-numeric types.
    Boolean  booleanVal;
    Char     charVal;
    DateTime datetimeVal;

// Numeric types.
    Byte     byteVal;
    Int16    int16Val;
    Int32    int32Val;
    Int64    int64Val;
    Decimal  decimalVal;
    Single   singleVal;
    Double   doubleVal;

// The following types are not CLS-compliant.
    SByte    sbyteVal;
    UInt16   uint16Val;
    UInt32   uint32Val;
    UInt64   uint64Val;
//
```

```
    Console.WriteLine(msg1, nl);

// Non-numeric types:
    Console.WriteLine(msg2, nl);
// DateTime
  // TryParse:
    // Assume current culture is en-US, and dates of the form: MMDDYYYY.
    success = DateTime.TryParse("7/4/2004 12:34:56", out datetimeVal);
    Show(success, "DateTime #1", datetimeVal.ToString());

    // Use fr-FR culture, and dates of the form: DDMMYYYY.
    ci = new CultureInfo("fr-FR");
    success = DateTime.TryParse("4/7/2004 12:34:56",
             ci, DateTimeStyles.None, out datetimeVal);
    Show(success, "DateTime #2", datetimeVal.ToString());

  // TryParseExact:
    // Use fr-FR culture. The format, "G", is short date and long time.
    success = DateTime.TryParseExact("04/07/2004 12:34:56", "G",
             ci, DateTimeStyles.None, out datetimeVal);
    Show(success, "DateTime #3", datetimeVal.ToString());

    // Assume en-US culture.
    string[] dateFormats = {"f", "F", "g", "G"};
    success = DateTime.TryParseExact("7/4/2004 12:34:56 PM",
             dateFormats, null, DateTimeStyles.None,
             out datetimeVal);
    Show(success, "DateTime #4", datetimeVal.ToString());

    Console.WriteLine();
// Boolean
    success = Boolean.TryParse("true", out booleanVal);
    Show(success, "Boolean", booleanVal.ToString());
// Char
    success = Char.TryParse("A", out charVal);
    Show(success, "Char", charVal.ToString());

// Numeric types:
    Console.WriteLine(msg3, nl);
// Byte
    success = Byte.TryParse("1", NumberStyles.Integer, null, out byteVal);
    Show(success, "Byte", byteVal.ToString());
// Int16
    success = Int16.TryParse("-2", NumberStyles.Integer, null, out
int16Val);
    Show(success, "Int16", int16Val.ToString());
// Int32
    success = Int32.TryParse("3", NumberStyles.Integer, null, out int32Val);
    Show(success, "Int32", int32Val.ToString());
// Int64
    success = Int64.TryParse("4", NumberStyles.Integer, null, out int64Val);
    Show(success, "Int64", int64Val.ToString());
// Decimal
    success = Decimal.TryParse("-5.5", NumberStyles.Number, null, out
decimalVal);
```

```
    Show(success, "Decimal", decimalVal.ToString());
// Single
    success = Single.TryParse("6.6",
             (NumberStyles.Float | NumberStyles.AllowThousands),
             null, out singleVal);
    Show(success, "Single", singleVal.ToString());
// Double
    success = Double.TryParse("-7",
             (NumberStyles.Float | NumberStyles.AllowThousands),
             null, out doubleVal);
    Show(success, "Double", doubleVal.ToString());

// Use the simple Double.TryParse overload, but specify an invalid value.

    success = Double.TryParse("abc", out doubleVal);
    Show(success, "Double #2", doubleVal.ToString());
//
    Console.WriteLine(msg4, nl);
// SByte
    success = SByte.TryParse("-8", NumberStyles.Integer, null, out
sbyteVal);
    Show(success, "SByte", sbyteVal.ToString());
// UInt16
    success = UInt16.TryParse("9", NumberStyles.Integer, null, out
uint16Val);
    Show(success, "UInt16", uint16Val.ToString());
// UInt32
    success = UInt32.TryParse("10", NumberStyles.Integer, null, out
uint32Val);
    Show(success, "UInt32", uint32Val.ToString());
// UInt64
    success = UInt64.TryParse("11", NumberStyles.Integer, null, out
uint64Val);
    Show(success, "UInt64", uint64Val.ToString());
    }

    protected static void Show(bool parseSuccess, string typeName,
                               string parseValue)
    {
    string msgSuccess = "Parse for {0} = {1}";
    string msgFailure = "** Parse for {0} failed. Invalid input.";
//
    if (parseSuccess == true)
        Console.WriteLine(msgSuccess, typeName, parseValue);
    else
        Console.WriteLine(msgFailure, typeName);
   }
}
/*
This example produces the following results:

This example demonstrates overloads of the TryParse method for
several base types, as well as the TryParseExact method for DateTime.

Non-numeric types:
```

```
Parse for DateTime #1 = 7/4/2004 12:34:56 PM
Parse for DateTime #2 = 7/4/2004 12:34:56 PM
Parse for DateTime #3 = 7/4/2004 12:34:56 PM
Parse for DateTime #4 = 7/4/2004 12:34:56 PM

Parse for Boolean = True
Parse for Char = A

Numeric types:

Parse for Byte = 1
Parse for Int16 = -2
Parse for Int32 = 3
Parse for Int64 = 4
Parse for Decimal = -5.5
Parse for Single = 6.6
Parse for Double = -7
** Parse for Double #2 failed. Invalid input.

The following types are not CLS-compliant:

Parse for SByte = -8
Parse for UInt16 = 9
Parse for UInt32 = 10
Parse for UInt64 = 11
*/
```

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Parse(String)
- ToString()
- Parsing Other Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Note This conversion is not supported. Attempting to do so throws an InvalidCastException.

```
C#

bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

This conversion is not supported. No value is returned.

Implements

ToBoolean(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The converted value of the current Char object.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

```
C#
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current Char object unchanged.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Note This conversion is not supported. Attempting to do so throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

DateTime

No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Note This conversion is not supported. Attempting to do so throws an InvalidCastException.

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Decimal

No value is returned.

Implements

ToDecimal(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Note This conversion is not supported. Attempting to do so throws an InvalidCastException.

C#

```
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Double

No value is returned.

Implements

ToDouble(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The converted value of the current Char object.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The converted value of the current Char object.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The converted value of the current Char object.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The converted value of the current Char object.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToSByte(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Note This conversion is not supported. Attempting to do so throws an InvalidCastException.

C#

```
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Single

No value is returned.

Implements

ToSingle(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

A Type object.

provider IFormatProvider

An IFormatProvider object.

Returns

Object

An object of the specified type.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

The value of the current Char object cannot be converted to the type specified by the `type` parameter.

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to call the ChangeType(Object, Type, IFormatProvider) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider object. (Specify `null` because the `provider` parameter is ignored.)

Returns

UInt16

The converted value of the current Char object.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt16(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider object. (Specify `null` because the `provider` parameter is ignored.)

Returns

UInt32

The converted value of the current Char object.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt32(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider object. (Specify `null` because the `provider` parameter is ignored.)

Returns

UInt64

The converted value of the current Char object.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Char instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt64(Char) method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IFormattable.ToString(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Formats the value of the current instance using the specified format.

C#

```
string IFormattable.ToString (string format, IFormatProvider
formatProvider);
```

Parameters

`format` String

The format to use. -or- A `null` reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`formatProvider` IFormatProvider

The provider to use to format the value. -or- A `null` reference (`Nothing` in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IParsable<Char>.Parse(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a string into a value.

```
C#
static char IParsable<char>.Parse (string s, IFormatProvider provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Char

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IParsable<Char>.TryParse(String, IFormatProvider, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IParsable<char>.TryParse (string s, IFormatProvider provider,
out char result);
```

Parameters

`s` String

`provider` IFormatProvider

`result` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance into the provided span of characters.

C#

```
bool ISpanFormattable.TryFormat (Span<char> destination, out int
charsWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.ISpanParsable<Char>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a span of characters into a value.

C#

```
static char ISpanParsable<char>.Parse (ReadOnlySpan<char> s, IFormatProvider
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Char

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.ISpanParsable<Char>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool ISpanParsable<char>.TryParse (ReadOnlySpan<char> s,
IFormatProvider provider, out char result);
```

Parameters

`s` ReadOnlySpan<Char>

`provider` IFormatProvider

`result` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.IUtf8SpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
bool IUtf8SpanFormattable.TryFormat (Span<byte> utf8Destination, out int
bytesWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.IAdditionOperators<Char,Char,Char>.Addition(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static char IAdditionOperators<char,char,char>.operator + (char left, char right);
```

Parameters

`left` Char

The value to which `right` is added.

`right` Char

The value that's added to `left`.

Returns

Char

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IAdditionOperators<Char,Char,Char>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static char IAdditionOperators<char,char,char>.op_CheckedAddition (char
left, char right);
```

Parameters

`left` Char

The value to which `right` is added.

`right` Char

The value that's added to `left`.

Returns

Char

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IAdditiveIdentity<Char,Char>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static char
System.Numerics.IAdditiveIdentity<System.Char,System.Char>.AdditiveIdentity
{ get; }
```

Property Value

Char

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.IBinaryInteger<Char>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<char>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.IBinaryInteger<Char>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<char>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.IBinaryInteger<Char>.LeadingZeroCount(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zero bits in a value.

C#

```
static char IBinaryInteger<char>.LeadingZeroCount (char value);
```

Parameters

`value` Char

The value whose leading zero bits are to be counted.

Returns

Char

The number of leading zero bits in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.PopCount(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

```
C#
static char IBinaryInteger<char>.PopCount (char value);
```

Parameters

`value` Char

The value whose set bits are to be counted.

Returns

Char

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.RotateLeft(Char, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
static char IBinaryInteger<char>.RotateLeft (char value, int rotateAmount);
```

Parameters

`value` Char

The value that's rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Char

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.RotateRight(Char, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
static char IBinaryInteger<char>.RotateRight (char value, int rotateAmount);
```

Parameters

`value` Char

The value that's rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Char

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.TrailingZeroCount(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zero bits in a value.

C#

```
static char IBinaryInteger<char>.TrailingZeroCount (char value);
```

Parameters

`value` Char

The value whose trailing zero bits are to be counted.

Returns

Char

The number of trailing zero bits in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<char>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out char value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.IBinaryInteger<Char>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<char>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out char value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Char.IBinaryInteger<Char>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<char>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryInteger<Char>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<char>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryNumber<Char>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static char System.Numerics.IBinaryNumber<System.Char>.AllBitsSet { get; }
```

Property Value

Char

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryNumber<Char>.IsPow2(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

```
C#
static bool IBinaryNumber<char>.IsPow2 (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBinaryNumber<Char>.Log2(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

```
C#
static char IBinaryNumber<char>.Log2 (char value);
```

Parameters

`value` Char

The value whose log2 is to be computed.

Returns

Char

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBitwise Operators<Char,Char,Char>.Bitwise And(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static char IBitwiseOperators<char,char,char>.operator & (char left, char
right);
```

Parameters

`left` Char

The value to `and` with `right`.

`right` Char

The value to `and` with `left`.

Returns

Char

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBitwiseOperators<Char,Char,Char>.BitwiseOr(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static char IBitwiseOperators<char,char,char>.operator | (char left, char
right);
```

Parameters

`left` Char

The value to `or` with `right`.

`right` Char

The value to `or` with `left`.

Returns

Char

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBitwise Operators<Char,Char,Char>.Exclusive Or(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static char IBitwiseOperators<char,char,char>.operator ^ (char left, char
right);
```

Parameters

`left` Char

The value to xor with `right`.

`right` Char

The value to xor with `left`.

Returns

Char

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IBitwise Operators<Char,Char,Char>.Ones Complement(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static char IBitwiseOperators<char,char,char>.operator ~ (char value);
```

Parameters

`value` Char

The value for which to compute the ones-complement.

Returns

Char

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IComparisonOperators<Char,Char,Boolean>.GreaterThanOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<char,char,bool>.operator > (char left, char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IComparisonOperators<Char,Char,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<char,char,bool>.operator >= (char left,
char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IComparisonOperators<Char,Char,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<char,char,bool>.operator < (char left, char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IComparisonOperators<Char,Char,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<char,char,bool>.operator <= (char left,
char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IDecrementOperators<Char>.CheckedDecrement(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static char IDecrementOperators<char>.op_CheckedDecrement (char value);
```

Parameters

`value` Char

The value to decrement.

Returns

Char

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IDecrementOperators<Char>.Decrement(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static char IDecrementOperators<char>.operator -- (char value);
```

Parameters

`value` Char

The value to decrement.

Returns

Char

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IDivisionOperators<Char,Char,Char>.Division(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static char IDivisionOperators<char,char,char>.operator / (char left, char
right);
```

Parameters

`left` Char

The value that `right` divides.

`right` Char

The value that divides `left`.

Returns

Char

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IEqualityOperators<Char,Char,Boolean>.Equality(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<char,char,bool>.operator == (char left, char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IEqualityOperators<Char,Char,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<char,char,bool>.operator != (char left, char right);
```

Parameters

`left` Char

The value to compare with `right`.

`right` Char

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IIncrementOperators<Char>.CheckedIncrement(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static char IIncrementOperators<char>.op_CheckedIncrement (char value);
```

Parameters

`value` Char

The value to increment.

Returns

Char

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IIncrementOperators<Char>.Increment(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static char IIncrementOperators<char>.operator ++ (char value);
```

Parameters

`value` Char

The value to increment.

Returns

Char

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IMinMaxValue<Char>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static char System.Numerics.IMinMaxValue<System.Char>.MaxValue { get; }
```

Property Value

Char

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IMinMaxValue<Char>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static char System.Numerics.IMinMaxValue<System.Char>.MinValue { get; }
```

Property Value

Char

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IModulusOperators<Char,Char,Char>.Modulus(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static char IModulusOperators<char,char,char>.operator % (char left, char
right);
```

Parameters

`left` Char

The value that `right` divides.

`right` Char

The value that divides `left`.

Returns

Char

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IMultiplicativeIdentity<Char,Char>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static char
System.Numerics.IMultiplicativeIdentity<System.Char,System.Char>.MultiplicativeIdentity { get; }
```

Property Value

Char

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.IMultiplyOperators<Char,Char,Char>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static char IMultiplyOperators<char,char,char>.op_CheckedMultiply (char
left, char right);
```

Parameters

left Char

The value that `right` multiplies.

right Char

The value that multiplies `left`.

Returns

Char

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IMultiplyOperators<Char,Char,Char>.Multiply(Char, Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static char IMultiplyOperators<char,char,char>.operator * (char left, char
right);
```

Parameters

left Char

The value that `right` multiplies.

right Char

The value that multiplies `left`.

Returns

Char

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.Abs(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static char INumberBase<char>.Abs (char value);
```

Parameters

`value` Char

The value for which to get its absolute.

Returns

Char

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsCanonical(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<char>.IsCanonical (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsComplexNumber(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<char>.IsComplexNumber (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsEvenInteger(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
static bool INumberBase<char>.IsEvenInteger (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsFinite(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<char>.IsFinite (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsImaginaryNumber(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<char>.IsImaginaryNumber (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsInfinity(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<char>.IsInfinity (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsInteger(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<char>.IsInteger (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsNaN(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<char>.IsNaN (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsNegative(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<char>.IsNegative (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsNegativeInfinity(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<char>.IsNegativeInfinity (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsNormal(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<char>.IsNormal (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsOddInteger(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
static bool INumberBase<char>.IsOddInteger (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsPositive(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<char>.IsPositive (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsPositiveInfinity(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<char>.IsPositiveInfinity (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsRealNumber(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<char>.IsRealNumber (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsSubnormal(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<char>.IsSubnormal (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.IsZero(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<char>.IsZero (char value);
```

Parameters

`value` Char

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.MaxMagnitude(Char, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static char INumberBase<char>.MaxMagnitude (char x, char y);
```

Parameters

x Char

The value to compare with `y`.

y Char

The value to compare with `x`.

Returns

Char

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.Max MagnitudeNumber(Char, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static char INumberBase<char>.MaxMagnitudeNumber (char x, char y);
```

Parameters

x Char

The value to compare with `y`.

y Char

The value to compare with `x`.

Returns

Char

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.MinMagnitude(Char, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
static char INumberBase<char>.MinMagnitude (char x, char y);
```

Parameters

x Char

The value to compare with `y`.

y Char

The value to compare with `x`.

Returns

Char

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.MinMagnitudeNumber(Char, Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

C#

```
static char INumberBase<char>.MinMagnitudeNumber (char x, char y);
```

Parameters

x Char

The value to compare with `y`.

y Char

The value to compare with `x`.

Returns

Char

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static char System.Numerics.INumberBase<System.Char>.One { get; }
```

Property Value

Char

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

INumberBase<Char>.Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
INumberBase<Char>.Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.

INumberBase<Char>.Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Parses a span of characters into a value.

C#

```
static char INumberBase<char>.Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Char

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

INumberBase<Char>.Parse(String, NumberStyles, IFormatProvider)

Parses a string into a value.

C#

```
static char INumberBase<char>.Parse (string s,
System.Globalization.NumberStyles style, IFormatProvider provider);
```

Parameters

`s` String

The string to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Char

The result of parsing `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Char>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<char>.TryConvertFromChecked<TOther> (TOther value,
out char result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<char>.TryConvertFromSaturating<TOther> (TOther
value, out char result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<char>.TryConvertFromTruncating<TOther> (TOther
value, out char result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Char

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<char>.TryConvertToChecked<TOther> (char value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Char

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Char.INumberBase<Char>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<char>.TryConvertToSaturating<TOther> (char value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Char

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<char>.TryConvertToTruncating<TOther> (char value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Char

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

INumberBase<Char>.TryParse(String, NumberStyles, IFormatProvider, Char)
INumberBase<Char>.TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Char)

INumberBase<Char>.TryParse(String, NumberStyles, IFormatProvider, Char)

C#

```
static bool INumberBase<char>.TryParse (string s,
System.Globalization.NumberStyles style, IFormatProvider provider, out
char result);
```

Parameters

`s` String

`style` NumberStyles

`provider` IFormatProvider

`result` Char

Returns

Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

INumberBase<Char>.TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Char)

C#

```
static bool INumberBase<char>.TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider provider, out
char result);
```

Parameters

`s` ReadOnlySpan<Char>

`style` NumberStyles

`provider` IFormatProvider

`result` Char

Returns

Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.INumberBase<Char>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static char System.Numerics.INumberBase<System.Char>.Zero { get; }
```

Property Value

Char

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Char.IShift Operators<Char,Int32,Char>.Left Shift(Char, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

C#

```
static char IShiftOperators<char,int,char>.operator << (char value, int
shiftAmount);
```

Parameters

`value` Char

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Char

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IShift Operators<Char,Int32,Char>.Right Shift(Char, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static char IShiftOperators<char,int,char>.operator >> (char value, int
shiftAmount);
```

Parameters

`value` Char

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Char

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IShift Operators<Char,Int32,Char>.Unsigned RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static char IShiftOperators<char,int,char>.op_UnsignedRightShift (char
value, int shiftAmount);
```

Parameters

`value` Char

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Char

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.ISubtractionOperators<Char,Char,Char>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static char ISubtractionOperators<char,char,char>.op_CheckedSubtraction
(char left, char right);
```

Parameters

`left` Char

The value from which `right` is subtracted.

`right` Char

The value that's subtracted from `left`.

Returns

Char

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.ISubtraction Operators<Char,Char,Char>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static char ISubtractionOperators<char,char,char>.operator - (char left,
char right);
```

Parameters

`left` Char

The value from which `right` is subtracted.

`right` Char

The value that's subtracted from `left`.

Returns

Char

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IUnaryNegationOperators<Char,Char>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static char IUnaryNegationOperators<char,char>.op_CheckedUnaryNegation (char
value);
```

Parameters

`value` Char

The value for which to compute the unary negation.

Returns

Char

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IUnaryNegationOperators<Char,Char>.UnaryNegation(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static char IUnaryNegationOperators<char,char>.operator - (char value);
```

Parameters

`value` Char

The value for which to compute the unary negation.

Returns

Char

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Char.IUnaryPlusOperators<Char,Char>.UnaryPlus(Char) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static char IUnaryPlusOperators<char,char>.operator + (char value);
```

Parameters

`value` Char

The value for which to compute the unary plus.

Returns

Char

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

CharEnumerator Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Supports iterating over a String object and reading its individual characters. This class cannot be inherited.

C#

```
public sealed class CharEnumerator : ICloneable,
System.Collections.Generic.IEnumerator<char>
```

Inheritance Object → CharEnumerator

Implements IEnumerator<Char> , IEnumerator , ICloneable , IDisposable

Examples

The following example uses the CharEnumerator class to enumerate the individual characters in a string. It instantiates a CharEnumerator object by calling the String.GetEnumerator method, moves from one character to the next by calling the MoveNext method, and displays the current character by retrieving the value of the Current property.

C#

```
string title = "A Tale of Two Cities";
CharEnumerator chEnum = title.GetEnumerator();
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
string outputLine3 = null;

while (chEnum.MoveNext())
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += chEnum.Current + " ";
   ctr++;
```

```
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
//                           1                   2
//       1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//       A   T a l e   o f   T w o   C i t i e s
```

Note, however, that the same operation can be performed somewhat more intuitively by using `foreach` (in C#) or `For Each` (in Visual Basic), as the following example shows.

C#

```
string title = "A Tale of Two Cities";
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
string outputLine3 = null;

foreach (char ch in title)
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += ch + " ";
   ctr++;
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
//                           1                   2
//       1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//       A   T a l e   o f   T w o   C i t i e s
```

Remarks

A CharEnumerator provides read-only access to the characters in a referenced String object. For example, the `foreach` statement of the Microsoft Visual Basic and C# programming languages, which iterates through the elements of a collection, retrieves a

CharEnumerator from a String object in order to iterate through the characters in that object.

> **Important**
>
> The `CharEnumerator` class enumerates individual 16-bit **Char** instances. It does not consider graphemes (that is, a character followed by one or more combiding characters) or surrogate pairs (that is, characters outside the Unicode Basic Multilingual Plane) as single characters. For an enumerator that handles these types of characters as a single unit, use the **StringInfo** class.

There is no public constructor for CharEnumerator. Instead, call a String object's GetEnumerator method to obtain a CharEnumerator that is initialized to reference the string.

A CharEnumerator maintains an internal index to the characters in the string the CharEnumerator references. The state of the index is invalid when it references a character position logically before the first character or after the last character in the string, and valid when it references a character within the string. The index is initialized to a position logically before the first character, and is set to a position after the last character when the iteration is complete. An exception is thrown if you attempt to access a character while the index is invalid.

The MoveNext method increments the index by one, so the first and subsequent characters are accessed in turn. The Reset method sets the index to a position logically before the first character. The Current property retrieves the character currently referenced by index. The Clone method creates a copy of the CharEnumerator.

> **Note**
>
> Several independent instances of **CharEnumerator** across one or more threads can have access to a single instance of **String**. This class is implemented to support the **IEnumerator** interface. For more information regarding the use of an enumerator, see the **IEnumerator** topic.

Properties

Expand table

Current	Gets the currently referenced character in the string enumerated by this CharEnumerator object.

Methods

Expand table

Clone()	Creates a copy of the current CharEnumerator object.
Dispose()	Releases all resources used by the current instance of the CharEnumerator class.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
MoveNext()	Increments the internal index of the current CharEnumerator object to the next character of the enumerated string.
Reset()	Initializes the index to a position logically before the first character of the enumerated string.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IEnumerator.Current	Gets the currently referenced character in the string enumerated by this CharEnumerator object. For a description of this member, see Current.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- String
- IEnumerator
- IEnumerable
- ICollection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CharEnumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the currently referenced character in the string enumerated by this CharEnumerator object.

```
C#
public char Current { get; }
```

Property Value

Char

The Unicode character currently referenced by this CharEnumerator object.

Implements

Current

Exceptions

InvalidOperationException

The index is invalid; that is, it is before the first or after the last character of the enumerated string.

Examples

The following example uses the CharEnumerator class to enumerate the individual characters in a string. It instantiates a CharEnumerator object by calling the String.GetEnumerator method, moves from one character to the next by calling the MoveNext method, and displays the current character by retrieving the value of the Current property.

```
C#
```

```
string title = "A Tale of Two Cities";
CharEnumerator chEnum = title.GetEnumerator();
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
string outputLine3 = null;

while (chEnum.MoveNext())
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += chEnum.Current + " ";
   ctr++;
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
//                           1                   2
//       1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//       A   T a l e   o f   T w o   C i t i e s
```

Note, however, that the same operation can be performed somewhat more intuitively by using `foreach` (in C#) or `For Each` (in Visual Basic), as the following example shows.

C#

```
string title = "A Tale of Two Cities";
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
string outputLine3 = null;

foreach (char ch in title)
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += ch + " ";
   ctr++;
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
```

```
//                        1                   2
//      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//      A   T a l e   o f   T w o   C i t i e s
```

Remarks

The CharEnumerator class maintains an internal index to the enumerated string, and the Current property returns the character that is currently referenced by the index. This property should be invoked only when the index is valid; otherwise, an exception is thrown.

The index is always invalid for an empty string ("""). The index is also invalid after the String.GetEnumerator or Reset method is called. After either of these methods is called, invoke the MoveNext method to adjust the index to the first character in the enumerated string. The index is valid whenever the MoveNext method returns `true`.

Current does not move the index, and consecutive calls to Current return the same character until MoveNext, Reset, or String.GetEnumerator is called.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CharEnumerator.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a copy of the current CharEnumerator object.

C#

```
public object Clone ();
```

Returns

Object

An Object that is a copy of the current CharEnumerator object.

Implements

Clone()

Remarks

The return value is a copy of this instance of CharEnumerator and its current state. This is useful for saving your state while iterating through a String object.

For example, suppose your application uses an original instance of CharEnumerator to iterate through each character in a String. When some unique character is encountered, your application pauses processing and invokes the Clone method. In effect, this saves the CharEnumerator object's index in the String.

Your application uses the clone to navigate to another part of the `String` to perform some auxiliary processing. The side-effect of this navigation is the clone loses track of the position where processing stopped. However, when the auxiliary processing is complete, your application discards the clone and uses the original CharEnumerator instance to resume working on the String where the original processing stopped.

> **Note**
>
> This method is implemented to support the **ICloneable** interface.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CharEnumerator.Dispose Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Releases all resources used by the current instance of the CharEnumerator class.

C#

```
public void Dispose ();
```

Implements

Dispose()

Remarks

Call Dispose when you are finished using the CharEnumerator. The Dispose method leaves the CharEnumerator in an unusable state. After calling Dispose, you must release all references to the CharEnumerator so the garbage collector can reclaim the memory that the CharEnumerator was occupying.

For more information, see Cleaning Up Unmanaged Resources and Implementing a Dispose Method.

> ⓘ **Note**
>
> Always call **Dispose** before you release your last reference to the **CharEnumerator**. Otherwise, the resources it is using will not be freed until the garbage collector calls the **CharEnumerator** object's `Finalize` method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

CharEnumerator.MoveNext Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments the internal index of the current CharEnumerator object to the next character of the enumerated string.

```
C#

public bool MoveNext ();
```

Returns

Boolean

`true` if the index is successfully incremented and within the enumerated string; otherwise, `false`.

Implements

MoveNext()

Examples

The following example uses the CharEnumerator class to enumerate the individual characters in a string. It instantiates a CharEnumerator object by calling the String.GetEnumerator method, moves from one character to the next by calling the MoveNext method, and displays the current character by retrieving the value of the Current property.

```
C#

string title = "A Tale of Two Cities";
CharEnumerator chEnum = title.GetEnumerator();
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
```

```
string outputLine3 = null;

while (chEnum.MoveNext())
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += chEnum.Current + " ";
   ctr++;
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
//                           1                   2
//       1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//       A   T a l e   o f   T w o   C i t i e s
```

Note, however, that the same operation can be performed somewhat more intuitively by using `foreach` (in C#) or `For Each` (in Visual Basic), as the following example shows.

C#

```
string title = "A Tale of Two Cities";
int ctr = 1;
string outputLine1 = null;
string outputLine2 = null;
string outputLine3 = null;

foreach (char ch in title)
{
   outputLine1 += ctr < 10 || ctr % 10 != 0 ? "  " : (ctr / 10) + " ";
   outputLine2 += (ctr % 10) + " ";
   outputLine3 += ch + " ";
   ctr++;
}

Console.WriteLine("The length of the string is {0} characters:",
                  title.Length);
Console.WriteLine(outputLine1);
Console.WriteLine(outputLine2);
Console.WriteLine(outputLine3);
// The example displays the following output to the console:
//       The length of the string is 20 characters:
//                           1                   2
//       1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//       A   T a l e   o f   T w o   C i t i e s
```

Remarks

The CharEnumerator class maintains an internal index to the enumerated string, and the MoveNext method increments the index by one. Call MoveNext after calling GetEnumerator or Reset to increment the current character position to the first character in the enumerated string. Check that the return value is `true` to determine that the current character position is valid.

If the index is already beyond the last character of the enumerated string, the index is not changed and `false` is returned.

Notice that if the enumerated string is empty ("""), the state of the CharEnumerator is always invalid. This is because the internal index for the CharEnumerator is initially before the first character of the enumerated string and is therefore invalid. MoveNext logically sets the index after the last (nonexistent) character of the enumerated string which is also invalid.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CharEnumerator.Reset Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes the index to a position logically before the first character of the enumerated string.

C#

```
public void Reset ();
```

Implements

Reset()

Remarks

The CharEnumerator class maintains an internal index to the enumerated string, and the Reset method sets the index to the invalid state.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

CharEnumerator.IEnumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the currently referenced character in the string enumerated by this CharEnumerator object. For a description of this member, see Current.

C#

```
object? System.Collections.IEnumerator.Current { get; }
```

Property Value

Object

The boxed Unicode character currently referenced by this CharEnumerator object.

Implements

Current

Exceptions

InvalidOperationException

Enumeration has not started.

-or-

Enumeration has ended.

Remarks

This member is an explicit interface member implementation. It can be used only when the CharEnumerator instance is cast to an IEnumerator interface.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CLSCompliantAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a program element is compliant with the Common Language Specification (CLS). This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.All, AllowMultiple=false,
Inherited=true)]
public sealed class CLSCompliantAttribute : Attribute
```

Inheritance Object → Attribute → CLSCompliantAttribute

Attributes AttributeUsageAttribute

Examples

The following example applies a CLSCompliantAttribute to the entire assembly.

C#

```
using System;
    [assembly: CLSCompliant(true)]
```

The following declaration generates a CLS-compliance warning because the type `UInt32` is not specified in the CLS.

C#

```
public int SetValue(UInt32 value);
```

If the declaration is marked with a CLSCompliantAttribute, no compiler warning or error is generated.

C#

```
[CLSCompliant(false)]
 public int SetValue(UInt32 value);
```

Remarks

The CLSCompliantAttribute attribute is used to indicate whether a particular program element complies with the Common Language Specification (CLS), which defines the features that any language that targets .NET must support. CLS compliance is primarily of concern to library developers who want to ensure that their libraries are accessible in any language that targets .NET. For more information, See Language Independence and Language-Independent Components.

You can apply the CLSCompliantAttribute attribute to the following program elements: assembly, module, class, struct, enum, constructor, method, property, field, event, interface, delegate, parameter, and return value. However, the notion of CLS compliance is only meaningful for assemblies, modules, types, and members of types, not parts of a member signature. Consequently, CLSCompliantAttribute is ignored when applied to parameter or return value program elements.

If no CLSCompliantAttribute is applied to a program element, then by default:

- The assembly is not CLS-compliant.
- The type is CLS-compliant only if its enclosing type or assembly is CLS-compliant.
- The member of a type is CLS-compliant only if the type is CLS-compliant.

If an assembly is marked as CLS-compliant, any publicly exposed type in the assembly that is not CLS-compliant must be marked with CLSCompliantAttribute using a `false` argument. Similarly, if a class is marked as CLS-compliant, you must individually mark all members that are not CLS-compliant. All non-compliant members must provide corresponding CLS-compliant alternatives.

Attributes that are applied to assemblies or modules must occur after the C# `using` (`Imports` in Visual Basic) clauses and before the code.

For more information about using attributes, see Attributes.

ⓘ **Note**

The current Microsoft Visual Basic compiler intentionally does not generate a CLS-compliance warning, however, a future release of the compiler will issue that warning.

Constructors

Expand table

CLSCompliant Attribute(Boolean)	Initializes an instance of the CLSCompliantAttribute class with a Boolean value indicating whether the indicated program element is CLS-compliant.

Properties

Expand table

Is Compliant	Gets the Boolean value indicating whether the indicated program element is CLS-compliant.
TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefault Attribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object.

	(Inherited from Attribute)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute
- Language Independence and Language-Independent Components
- Extending Metadata Using Attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

CLSCompliantAttribute(Boolean) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes an instance of the CLSCompliantAttribute class with a Boolean value indicating whether the indicated program element is CLS-compliant.

C#

```
public CLSCompliantAttribute (bool isCompliant);
```

Parameters

`isCompliant` Boolean

`true` if CLS-compliant; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

CLSCompliantAttribute.IsCompliant Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Boolean value indicating whether the indicated program element is CLS-compliant.

C#

```
public bool IsCompliant { get; }
```

Property Value

Boolean

`true` if the program element is CLS-compliant; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Comparison<T> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that compares two objects of the same type.

```
C#
public delegate int Comparison<in T>(T x, T y);
```

Type Parameters

T

The type of the objects to compare.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

x T

The first object to compare.

y T

The second object to compare.

Return Value

Int32

A signed integer that indicates the relative values of `x` and `y`, as shown in the following table.

Expand table

Value	Meaning
Less than 0	`x` is less than `y`.
0	`x` equals `y`.
Greater than 0	`x` is greater than `y`.

Examples

The following code example demonstrates the use of the Comparison<T> delegate with the Sort(Comparison<T>) method overload.

The code example defines an alternative comparison method for strings, named `CompareDinosByLength`. This method works as follows: First, the comparands are tested for `null`, and a null reference is treated as less than a non-null. Second, the string lengths are compared, and the longer string is deemed to be greater. Third, if the lengths are equal, ordinary string comparison is used.

A List<T> of strings is created and populated with four strings, in no particular order. The list also includes an empty string and a null reference. The list is displayed, sorted using a Comparison<T> generic delegate representing the `CompareDinosByLength` method, and displayed again.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
    private static int CompareDinosByLength(string x, string y)
    {
        if (x == null)
        {
            if (y == null)
            {
                // If x is null and y is null, they're
                // equal.
                return 0;
            }
            else
            {
                // If x is null and y is not null, y
                // is greater.
                return -1;
            }
        }
```

```
        else
        {
            // If x is not null...
            //
            if (y == null)
                // ...and y is null, x is greater.
            {
                return 1;
            }
            else
            {
                // ...and y is not null, compare the
                // lengths of the two strings.
                //
                int retval = x.Length.CompareTo(y.Length);

                if (retval != 0)
                {
                    // If the strings are not of equal length,
                    // the longer string is greater.
                    //
                    return retval;
                }
                else
                {
                    // If the strings are of equal length,
                    // sort them with ordinary string comparison.
                    //
                    return x.CompareTo(y);
                }
            }
        }
    }

    public static void Main()
    {
        List<string> dinosaurs = new List<string>();
        dinosaurs.Add("Pachycephalosaurus");
        dinosaurs.Add("Amargasaurus");
        dinosaurs.Add("");
        dinosaurs.Add(null);
        dinosaurs.Add("Mamenchisaurus");
        dinosaurs.Add("Deinonychus");
        Display(dinosaurs);

        Console.WriteLine("\nSort with generic Comparison<string>
delegate:");
        dinosaurs.Sort(CompareDinosByLength);
        Display(dinosaurs);
    }

    private static void Display(List<string> list)
    {
        Console.WriteLine();
        foreach( string s in list )
```

```
        {
            if (s == null)
                Console.WriteLine("(null)");
            else
                Console.WriteLine("\"{0}\"", s);
        }
    }
}

/* This code example produces the following output:

"Pachycephalosaurus"
"Amargasaurus"
""
(null)
"Mamenchisaurus"
"Deinonychus"

Sort with generic Comparison<string> delegate:

(null)
""
"Deinonychus"
"Amargasaurus"
"Mamenchisaurus"
"Pachycephalosaurus"
 */
```

The following example uses the Comparison<T> delegate to sort the elements of a collection of `CityInfo` objects. `CityInfo` is an application-defined class that contains information about a city and its population. The example defines three methods, `CompareByName`, `CompareByPopulation`, and `CompareByNames`, that offer three different ways of ordering the `CityInfo` objects. Each method is assigned to the `comparison` argument of the Array.Sort<T>(T[], Comparison<T>) method.

C#

```
using System;

public class CityInfo
{
   string cityName;
   string countryName;
   int pop2010;

   public CityInfo(string name, string country, int pop2010)
   {
      this.cityName = name;
      this.countryName = country;
      this.pop2010 = pop2010;
   }
```

```
   public string City
   { get { return this.cityName; } }

   public string Country
   { get { return this.countryName; } }

   public int Population
   { get { return this.pop2010; } }

   public static int CompareByName(CityInfo city1, CityInfo city2)
   {
      return String.Compare(city1.City, city2.City);
   }

   public static int CompareByPopulation(CityInfo city1, CityInfo city2)
   {
      return city1.Population.CompareTo(city2.Population);
   }

   public static int CompareByNames(CityInfo city1, CityInfo city2)
   {
      return String.Compare(city1.Country + city1.City, city2.Country +
city2.City);
   }
}

public class Example
{
   public static void Main()
   {
      CityInfo NYC = new CityInfo("New York City", "United States of
America", 8175133 );
      CityInfo Det = new CityInfo("Detroit", "United States of America",
713777);
      CityInfo Paris = new CityInfo("Paris", "France",  2193031);
      CityInfo[] cities = { NYC, Det, Paris };
      // Display ordered array.
      DisplayArray(cities);

      // Sort array by city name.
      Array.Sort(cities, CityInfo.CompareByName);
      DisplayArray(cities);

      // Sort array by population.
      Array.Sort(cities, CityInfo.CompareByPopulation);
      DisplayArray(cities);

      // Sort array by country + city name.
      Array.Sort(cities, CityInfo.CompareByNames);
      DisplayArray(cities);
   }

   private static void DisplayArray(CityInfo[] cities)
   {
```

```
      Console.WriteLine("{0,-20} {1,-25} {2,10}", "City", "Country",
"Population");
      foreach (var city in cities)
         Console.WriteLine("{0,-20} {1,-25} {2,10:N0}", city.City,
                           city.Country, city.Population);

      Console.WriteLine();
   }
}
// The example displays the following output:
//     City                 Country                   Population
//     New York City        United States of America   8,175,133
//     Detroit              United States of America     713,777
//     Paris                France                    2,193,031
//
//     City                 Country                   Population
//     Detroit              United States of America     713,777
//     New York City        United States of America   8,175,133
//     Paris                France                    2,193,031
//
//     City                 Country                   Population
//     Detroit              United States of America     713,777
//     Paris                France                    2,193,031
//     New York City        United States of America   8,175,133
//
//     City                 Country                   Population
//     Paris                France                    2,193,031
//     Detroit              United States of America     713,777
//     New York City        United States of America   8,175,133
```

Remarks

This delegate is used by the Sort<T>(T[], Comparison<T>) method overload of the Array class and the Sort(Comparison<T>) method overload of the List<T> class to sort the elements of an array or list.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Sort<T>(T[], Comparison<T>)
- Sort(Comparison<T>)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console Class

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Represents the standard input, output, and error streams for console applications. This class cannot be inherited.

C#

```
public static class Console
```

Inheritance Object → Console

Remarks

For more information about this API, see Supplemental API remarks for Console.

Properties

Expand table

BackgroundColor	Gets or sets the background color of the console.
BufferHeight	Gets or sets the height of the buffer area.
BufferWidth	Gets or sets the width of the buffer area.
CapsLock	Gets a value indicating whether the CAPS LOCK keyboard toggle is turned on or turned off.
CursorLeft	Gets or sets the column position of the cursor within the buffer area.
CursorSize	Gets or sets the height of the cursor within a character cell.
CursorTop	Gets or sets the row position of the cursor within the buffer area.
CursorVisible	Gets or sets a value indicating whether the cursor is visible.

Error	Gets the standard error output stream.
Foreground Color	Gets or sets the foreground color of the console.
In	Gets the standard input stream.
InputEncoding	Gets or sets the encoding the console uses to read input.
IsError Redirected	Gets a value that indicates whether the error output stream has been redirected from the standard error stream.
IsInput Redirected	Gets a value that indicates whether input has been redirected from the standard input stream.
IsOutput Redirected	Gets a value that indicates whether output has been redirected from the standard output stream.
KeyAvailable	Gets a value indicating whether a key press is available in the input stream.
LargestWindow Height	Gets the largest possible number of console window rows, based on the current font and screen resolution.
LargestWindow Width	Gets the largest possible number of console window columns, based on the current font and screen resolution.
NumberLock	Gets a value indicating whether the NUM LOCK keyboard toggle is turned on or turned off.
Out	Gets the standard output stream.
Output Encoding	Gets or sets the encoding the console uses to write output.
Title	Gets or sets the title to display in the console title bar.
TreatControl CAsInput	Gets or sets a value indicating whether the combination of the Control modifier key and C console key (Ctrl+C) is treated as ordinary input or as an interruption that is handled by the operating system.
WindowHeight	Gets or sets the height of the console window area.
WindowLeft	Gets or sets the leftmost position of the console window area relative to the screen buffer.
WindowTop	Gets or sets the top position of the console window area relative to the screen buffer.
WindowWidth	Gets or sets the width of the console window.

Methods

Expand table

Beep()	Plays the sound of a beep through the console speaker.
Beep(Int32, Int32)	Plays the sound of a beep of a specified frequency and duration through the console speaker.
Clear()	Clears the console buffer and corresponding console window of display information.
GetCursorPosition()	Gets the position of the cursor.
MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32)	Copies a specified source area of the screen buffer to a specified destination area.
MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32, Char, ConsoleColor, ConsoleColor)	Copies a specified source area of the screen buffer to a specified destination area.
OpenStandardError()	Acquires the standard error stream.
OpenStandardError(Int32)	Acquires the standard error stream, which is set to a specified buffer size.
OpenStandardInput()	Acquires the standard input stream.
OpenStandardInput(Int32)	Acquires the standard input stream, which is set to a specified buffer size.
OpenStandardOutput()	Acquires the standard output stream.
OpenStandardOutput(Int32)	Acquires the standard output stream, which is set to a specified buffer size.
Read()	Reads the next character from the standard input stream.
ReadKey()	Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.
ReadKey(Boolean)	Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.
ReadLine()	Reads the next line of characters from the standard input stream.
ResetColor()	Sets the foreground and background console colors to their defaults.

SetBufferSize(Int32, Int32)	Sets the height and width of the screen buffer area to the specified values.
SetCursorPosition(Int32, Int32)	Sets the position of the cursor.
SetError(TextWriter)	Sets the Error property to the specified TextWriter object.
SetIn(TextReader)	Sets the In property to the specified TextReader object.
SetOut(TextWriter)	Sets the Out property to target the TextWriter object.
SetWindowPosition(Int32, Int32)	Sets the position of the console window relative to the screen buffer.
SetWindowSize(Int32, Int32)	Sets the height and width of the console window to the specified values.
Write(Boolean)	Writes the text representation of the specified Boolean value to the standard output stream.
Write(Char)	Writes the specified Unicode character value to the standard output stream.
Write(Char[])	Writes the specified array of Unicode characters to the standard output stream.
Write(Char[], Int32, Int32)	Writes the specified subarray of Unicode characters to the standard output stream.
Write(Decimal)	Writes the text representation of the specified Decimal value to the standard output stream.
Write(Double)	Writes the text representation of the specified double-precision floating-point value to the standard output stream.
Write(Int32)	Writes the text representation of the specified 32-bit signed integer value to the standard output stream.
Write(Int64)	Writes the text representation of the specified 64-bit signed integer value to the standard output stream.
Write(Object)	Writes the text representation of the specified object to the standard output stream.
Write(Single)	Writes the text representation of the specified single-precision floating-point value to the standard output stream.
Write(String)	Writes the specified string value to the standard output stream.

Write(String, Object)	Writes the text representation of the specified object to the standard output stream using the specified format information.
Write(String, Object, Object)	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write(String, Object, Object, Object)	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write(String, Object[])	Writes the text representation of the specified array of objects to the standard output stream using the specified format information.
Write(UInt32)	Writes the text representation of the specified 32-bit unsigned integer value to the standard output stream.
Write(UInt64)	Writes the text representation of the specified 64-bit unsigned integer value to the standard output stream.
WriteLine()	Writes the current line terminator to the standard output stream.
WriteLine(Boolean)	Writes the text representation of the specified Boolean value, followed by the current line terminator, to the standard output stream.
WriteLine(Char)	Writes the specified Unicode character, followed by the current line terminator, value to the standard output stream.
WriteLine(Char[])	Writes the specified array of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine(Char[], Int32, Int32)	Writes the specified subarray of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine(Decimal)	Writes the text representation of the specified Decimal value, followed by the current line terminator, to the standard output stream.
WriteLine(Double)	Writes the text representation of the specified double-precision floating-point value, followed by the current line terminator, to the standard output stream.
WriteLine(Int32)	Writes the text representation of the specified 32-bit signed integer value, followed by the current line

	terminator, to the standard output stream.
WriteLine(Int64)	Writes the text representation of the specified 64-bit signed integer value, followed by the current line terminator, to the standard output stream.
WriteLine(Object)	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream.
WriteLine(Single)	Writes the text representation of the specified single-precision floating-point value, followed by the current line terminator, to the standard output stream.
WriteLine(String)	Writes the specified string value, followed by the current line terminator, to the standard output stream.
WriteLine(String, Object)	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(String, Object, Object)	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(String, Object, Object, Object)	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(String, Object[])	Writes the text representation of the specified array of objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(UInt32)	Writes the text representation of the specified 32-bit unsigned integer value, followed by the current line terminator, to the standard output stream.
WriteLine(UInt64)	Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the standard output stream.

Events

Expand table

CancelKeyPress	Occurs when the Control modifier key (Ctrl) and either the C console key (C) or the Break key are pressed simultaneously (Ctrl+C or Ctrl+Break).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Thread Safety

This type is thread safe.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.BackgroundColor Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the background color of the console.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static ConsoleColor BackgroundColor { get; set; }
```

Property Value

ConsoleColor

A value that specifies the background color of the console; that is, the color that appears behind each character. The default is black.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentException

The color specified in a set operation is not a valid member of ConsoleColor.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

The following example saves the values of the ConsoleColor enumeration to an array and stores the current values of the BackgroundColor and ForegroundColor properties to variables. It then changes the foreground color to each color in the ConsoleColor enumeration except to the color that matches the current background, and it changes the background color to each color in the ConsoleColor enumeration except to the color that matches the current foreground. (If the foreground color is the same as the background color, the text isn't visible.) Finally, it calls the ResetColor method to restore the original console colors.

C#

```
using System;

class Example
{
   public static void Main()
   {
      // Get an array with the values of ConsoleColor enumeration members.
      ConsoleColor[] colors = (ConsoleColor[])
ConsoleColor.GetValues(typeof(ConsoleColor));
      // Save the current background and foreground colors.
      ConsoleColor currentBackground = Console.BackgroundColor;
      ConsoleColor currentForeground = Console.ForegroundColor;

      // Display all foreground colors except the one that matches the
background.
      Console.WriteLine("All the foreground colors except {0}, the
background color:",
                        currentBackground);
      foreach (var color in colors) {
         if (color == currentBackground) continue;

         Console.ForegroundColor = color;
         Console.WriteLine("   The foreground color is {0}.", color);
      }
      Console.WriteLine();
      // Restore the foreground color.
      Console.ForegroundColor = currentForeground;

      // Display each background color except the one that matches the
current foreground color.
      Console.WriteLine("All the background colors except {0}, the
foreground color:",
                        currentForeground);
      foreach (var color in colors) {
         if (color == currentForeground) continue;

         Console.BackgroundColor = color;
         Console.WriteLine("   The background color is {0}.", color);
      }

      // Restore the original console colors.
```

```
        Console.ResetColor();
        Console.WriteLine("\nOriginal colors restored...");
    }
}
//The example displays output like the following:
//    All the foreground colors except DarkCyan, the background color:
//       The foreground color is Black.
//       The foreground color is DarkBlue.
//       The foreground color is DarkGreen.
//       The foreground color is DarkRed.
//       The foreground color is DarkMagenta.
//       The foreground color is DarkYellow.
//       The foreground color is Gray.
//       The foreground color is DarkGray.
//       The foreground color is Blue.
//       The foreground color is Green.
//       The foreground color is Cyan.
//       The foreground color is Red.
//       The foreground color is Magenta.
//       The foreground color is Yellow.
//       The foreground color is White.
//
//    All the background colors except White, the foreground color:
//       The background color is Black.
//       The background color is DarkBlue.
//       The background color is DarkGreen.
//       The background color is DarkCyan.
//       The background color is DarkRed.
//       The background color is DarkMagenta.
//       The background color is DarkYellow.
//       The background color is Gray.
//       The background color is DarkGray.
//       The background color is Blue.
//       The background color is Green.
//       The background color is Cyan.
//       The background color is Red.
//       The background color is Magenta.
//       The background color is Yellow.
//
//    Original colors restored...
```

Remarks

A change to the BackgroundColor property affects only output that is written to individual character cells after the background color is changed. To change the background color of the console window as a whole, set the BackgroundColor property and call the Clear method. The following example provides an illustration.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      WriteCharacterStrings(1, 26, true);
      Console.MoveBufferArea(0, Console.CursorTop - 10, 30, 1,
                             Console.CursorLeft, Console.CursorTop + 1);
      Console.CursorTop = Console.CursorTop + 3;
      Console.WriteLine("Press any key...");
      Console.ReadKey();

      Console.Clear();
      WriteCharacterStrings(1, 26, false);
   }

   private static void WriteCharacterStrings(int start, int end,
                                             bool changeColor)
   {
      for (int ctr = start; ctr <= end; ctr++) {
         if (changeColor)
            Console.BackgroundColor = (ConsoleColor) ((ctr - 1) % 16);

         Console.WriteLine(new String((char)(ctr + 64), 30));
      }
   }
}
```

A get operation for a Windows-based application, in which a console does not exist, returns ConsoleColor.Black. Unix systems don't provide any general mechanism to fetch the current console colors. Because of that, BackgroundColor returns `(ConsoleColor)-1` until it is set in explicit way (using the setter).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.BufferHeight Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the height of the buffer area.

C#

```
public static int BufferHeight {
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] get;
[System.Runtime.Versioning.SupportedOSPlatform("windows")] set; }
```

Property Value

Int32

The current height, in rows, of the buffer area.

Attributes UnsupportedOSPlatformAttribute, SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value in a set operation is less than or equal to zero.

-or-

The value in a set operation is greater than or equal to Int16.MaxValue.

-or-

The value in a set operation is less than WindowTop + WindowHeight.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the BufferHeight and BufferWidth properties. The example reports the dimensions of an operating system window set to a buffer size of 300 rows and 85 columns.

C#

```
// This example demonstrates the Console.BufferHeight and
//                               Console.BufferWidth properties.
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine("The current buffer height is {0} rows.",
                      Console.BufferHeight);
    Console.WriteLine("The current buffer width is {0} columns.",
                      Console.BufferWidth);
    }
}
/*
This example produces the following results:

The current buffer height is 300 rows.
The current buffer width is 85 columns.
*/
```

Remarks

This property defines the number of rows (or lines) stored in the buffer that is accessed by a console mode window. In contrast, the WindowHeight property defines the number of rows that are actually displayed in the console window at any particular time. If the number of rows actually written to the buffer exceeds the number of rows defined by the WindowHeight property, the window can be scrolled vertically so that it displays a contiguous number of rows that are equal to the WindowHeight property and are located anywhere in the buffer.

If a set operation decreases the value of the BufferHeight property, the uppermost lines are removed. For example, if the number of lines is reduced from 300 to 250, lines 0 through 49 are removed, and the existing lines 50 through 299 become lines 0 through 249.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.BufferWidth Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the width of the buffer area.

C#

```
public static int BufferWidth {
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] get;
[System.Runtime.Versioning.SupportedOSPlatform("windows")] set; }
```

Property Value

Int32

The current width, in columns, of the buffer area.

Attributes UnsupportedOSPlatformAttribute, SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value in a set operation is less than or equal to zero.

-or-

The value in a set operation is greater than or equal to Int16.MaxValue.

-or-

The value in a set operation is less than WindowLeft + WindowWidth.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the BufferHeight and BufferWidth properties. The example reports the dimensions of an operating system window set to a buffer size of 300 rows and 85 columns.

C#

```
// This example demonstrates the Console.BufferHeight and
//                                Console.BufferWidth properties.
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine("The current buffer height is {0} rows.",
                      Console.BufferHeight);
    Console.WriteLine("The current buffer width is {0} columns.",
                      Console.BufferWidth);
    }
}
/*
This example produces the following results:

The current buffer height is 300 rows.
The current buffer width is 85 columns.
*/
```

Remarks

If a set operation decreases the value of the BufferWidth property, the rightmost columns are removed. For example, if the number of columns is reduced from 80 to 60, columns 60 through 79 of each row are removed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.CapsLock Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value indicating whether the CAPS LOCK keyboard toggle is turned on or turned off.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static bool CapsLock { get; }
```

Property Value

Boolean

`true` if CAPS LOCK is turned on; `false` if CAPS LOCK is turned off.

Attributes SupportedOSPlatformAttribute

Exceptions

PlatformNotSupportedException

The get operation is invoked on an operating system other than Windows.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.CursorLeft Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the column position of the cursor within the buffer area.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int CursorLeft { get; set; }
```

Property Value

Int32

The current position, in columns, of the cursor.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value in a set operation is less than zero.

-or-

The value in a set operation is greater than or equal to BufferWidth.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

This example demonstrates the CursorLeft and CursorTop properties, and the SetCursorPosition and Clear methods. The example positions the cursor, which determines where the next write will occur, to draw a 5 character by 5 character rectangle using a combination of "+", "|", and "-" strings. Note that the rectangle could be drawn with fewer steps using a combination of other strings.

C#

```
// This example demonstrates the
//     Console.CursorLeft and
//     Console.CursorTop properties, and the
//     Console.SetCursorPosition and
//     Console.Clear methods.
using System;

class Sample
{
    protected static int origRow;
    protected static int origCol;

    protected static void WriteAt(string s, int x, int y)
    {
    try
        {
        Console.SetCursorPosition(origCol+x, origRow+y);
        Console.Write(s);
        }
    catch (ArgumentOutOfRangeException e)
        {
        Console.Clear();
        Console.WriteLine(e.Message);
        }
    }

    public static void Main()
    {
// Clear the screen, then save the top and left coordinates.
    Console.Clear();
    origRow = Console.CursorTop;
    origCol = Console.CursorLeft;

// Draw the left side of a 5x5 rectangle, from top to bottom.
    WriteAt("+", 0, 0);
    WriteAt("|", 0, 1);
    WriteAt("|", 0, 2);
    WriteAt("|", 0, 3);
    WriteAt("+", 0, 4);

// Draw the bottom side, from left to right.
    WriteAt("-", 1, 4); // shortcut: WriteAt("---", 1, 4)
```

```
    WriteAt("-", 2, 4); // ...
    WriteAt("-", 3, 4); // ...
    WriteAt("+", 4, 4);

// Draw the right side, from bottom to top.
    WriteAt("|", 4, 3);
    WriteAt("|", 4, 2);
    WriteAt("|", 4, 1);
    WriteAt("+", 4, 0);

// Draw the top side, from right to left.
    WriteAt("-", 3, 0); // shortcut: WriteAt("---", 1, 0)
    WriteAt("-", 2, 0); // ...
    WriteAt("-", 1, 0); // ...
//
    WriteAt("All done!", 0, 6);
    Console.WriteLine();
    }
}
/*
This example produces the following results:

+---+
|   |
|   |
|   |
+---+

All done!

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Console.CursorSize Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the height of the cursor within a character cell.

C#

```
public static int CursorSize {
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] get;
[System.Runtime.Versioning.SupportedOSPlatform("windows")] set; }
```

Property Value

Int32

The size of the cursor expressed as a percentage of the height of a character cell. The property value ranges from 1 to 100.

Attributes UnsupportedOSPlatformAttribute, SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value specified in a set operation is less than 1 or greater than 100.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the CursorSize property. The example increases the size of the cursor each time any console key is pressed, then restores the cursor to its original size before terminating.

C#

```
// This example demonstrates the Console.CursorSize property.
using System;

class Sample
{
    public static void Main()
    {
    string m0 = "This example increments the cursor size from 1% to
100%:\n";
    string m1 = "Cursor size = {0}%. (Press any key to continue...)";
    int[] sizes = {1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int saveCursorSize;
//
    saveCursorSize = Console.CursorSize;
    Console.WriteLine(m0);
    foreach (int size in sizes)
        {
        Console.CursorSize = size;
        Console.WriteLine(m1, size);
        Console.ReadKey();
        }
    Console.CursorSize = saveCursorSize;
    }
}
/*
This example produces the following results:

This example increments the cursor size from 1% to 100%:

Cursor size = 1%. (Press any key to continue...)
Cursor size = 10%. (Press any key to continue...)
Cursor size = 20%. (Press any key to continue...)
Cursor size = 30%. (Press any key to continue...)
Cursor size = 40%. (Press any key to continue...)
Cursor size = 50%. (Press any key to continue...)
Cursor size = 60%. (Press any key to continue...)
Cursor size = 70%. (Press any key to continue...)
Cursor size = 80%. (Press any key to continue...)
Cursor size = 90%. (Press any key to continue...)
Cursor size = 100%. (Press any key to continue...)

*/
```

Remarks

The cursor appearance varies, ranging from a horizontal line at the bottom of the cell when the property value is 1, to completely filling the cell when the property value is 100.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.CursorTop Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the row position of the cursor within the buffer area.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int CursorTop { get; set; }
```

Property Value

Int32

The current position, in rows, of the cursor.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value in a set operation is less than zero.

-or-

The value in a set operation is greater than or equal to BufferHeight.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

This example demonstrates the CursorLeft and CursorTop properties, and the SetCursorPosition and Clear methods. The example positions the cursor, which determines where the next write will occur, to draw a 5 character by 5 character rectangle using a combination of "+", "|", and "-" strings. Note that the rectangle could be drawn with fewer steps using a combination of other strings.

C#

```
// This example demonstrates the
//     Console.CursorLeft and
//     Console.CursorTop properties, and the
//     Console.SetCursorPosition and
//     Console.Clear methods.
using System;

class Sample
{
    protected static int origRow;
    protected static int origCol;

    protected static void WriteAt(string s, int x, int y)
    {
    try
        {
        Console.SetCursorPosition(origCol+x, origRow+y);
        Console.Write(s);
        }
    catch (ArgumentOutOfRangeException e)
        {
        Console.Clear();
        Console.WriteLine(e.Message);
        }
    }

    public static void Main()
    {
// Clear the screen, then save the top and left coordinates.
    Console.Clear();
    origRow = Console.CursorTop;
    origCol = Console.CursorLeft;

// Draw the left side of a 5x5 rectangle, from top to bottom.
    WriteAt("+", 0, 0);
    WriteAt("|", 0, 1);
    WriteAt("|", 0, 2);
    WriteAt("|", 0, 3);
    WriteAt("+", 0, 4);

// Draw the bottom side, from left to right.
    WriteAt("-", 1, 4); // shortcut: WriteAt("---", 1, 4)
```

```
    WriteAt("-", 2, 4); // ...
    WriteAt("-", 3, 4); // ...
    WriteAt("+", 4, 4);

// Draw the right side, from bottom to top.
    WriteAt("|", 4, 3);
    WriteAt("|", 4, 2);
    WriteAt("|", 4, 1);
    WriteAt("+", 4, 0);

// Draw the top side, from right to left.
    WriteAt("-", 3, 0); // shortcut: WriteAt("---", 1, 0)
    WriteAt("-", 2, 0); // ...
    WriteAt("-", 1, 0); // ...
//
    WriteAt("All done!", 0, 6);
    Console.WriteLine();
    }
}
/*
This example produces the following results:

+---+
|   |
|   |
|   |
+---+

All done!

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Console.CursorVisible Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets a value indicating whether the cursor is visible.

C#

```
public static bool CursorVisible {
[System.Runtime.Versioning.SupportedOSPlatform("windows")] get;
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] set; }
```

Property Value

Boolean

`true` if the cursor is visible; otherwise, `false`.

Attributes SupportedOSPlatformAttribute, UnsupportedOSPlatformAttribute

Exceptions

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The get operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the CursorVisible property. The example makes the cursor visible if the first column of input is a '+' character or invisible if the input is a '-' character.

C#

```
// This example demonstrates the Console.CursorVisible property.

using System;

class Sample
{
    public static void Main()
    {
    string m1 = "\nThe cursor is {0}.\nType any text then press Enter. " +
                "Type '+' in the first column to show \n" +
                "the cursor, '-' to hide the cursor, " +
                "or lowercase 'x' to quit:";
    string s;
    bool saveCursorVisibile;
    int  saveCursorSize;
//
    Console.CursorVisible = true; // Initialize the cursor to visible.
    saveCursorVisibile = Console.CursorVisible;
    saveCursorSize  = Console.CursorSize;
    Console.CursorSize = 100;     // Emphasize the cursor.

    while(true)
        {
        Console.WriteLine(m1,
                         ((Console.CursorVisible == true) ?
                           "VISIBLE" : "HIDDEN"));
        s = Console.ReadLine();
        if (String.IsNullOrEmpty(s) == false)
            if (s[0] == '+')
                Console.CursorVisible = true;
            else if (s[0] == '-')
                Console.CursorVisible = false;
            else if (s[0] == 'x')
                break;
        }
    Console.CursorVisible = saveCursorVisibile;
    Console.CursorSize    = saveCursorSize;
    }
}
/*
This example produces the following results. Note that these results
cannot depict cursor visibility. You must run the example to see the
cursor behavior:

The cursor is VISIBLE.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
```

```
The quick brown fox

The cursor is VISIBLE.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
-

The cursor is HIDDEN.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
jumps over

The cursor is HIDDEN.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
+

The cursor is VISIBLE.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
the lazy dog.

The cursor is VISIBLE.
Type any text then press Enter. Type '+' in the first column to show
the cursor, '-' to hide the cursor, or lowercase 'x' to quit:
x

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Console.Error Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the standard error output stream.

C#

```
public static System.IO.TextWriter Error { get; }
```

Property Value

TextWriter

A TextWriter that represents the standard error output stream.

Examples

The following example is a command line utility named ExpandTabs that replaces tab characters in a text file with four spaces, the value defined by the `tabSize` variable. It redirects the standard input and output streams to files, but uses the Error property to write the standard error stream to the console. It can be launched from the command line by supplying the name of the file that contains tab characters and the name of the output file.

C#

```
using System;
using System.IO;

public class ExpandTabs
{
    private const int tabSize = 4;
    private const string usageText = "Usage: EXPANDTABSEX inputfile.txt
outputfile.txt";

    public static void Main(string[] args)
    {
        StreamWriter writer = null;
```

```
        if (args.Length < 2) {
            Console.WriteLine(usageText);
            return;
        }

        try {
            writer = new StreamWriter(args[1]);
            Console.SetOut(writer);
            Console.SetIn(new StreamReader(args[0]));
        }
        catch(IOException e) {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return;
        }
        int i;
        while ((i = Console.Read()) != -1) {
            char c = (char)i;
            if (c == '\t')
                Console.Write("").PadRight(tabSize, ' '));
            else
                Console.Write(c);
        }
        writer.Close();
        // Recover the standard output stream so that a
        // completion message can be displayed.
        StreamWriter standardOutput = new
StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
        Console.WriteLine("EXPANDTABSEX has completed the processing of
{0}.", args[0]);
        return;
    }
}
```

The following example is a simple text file viewer that displays the contents of one or more text files to the console. If there are no command line arguments, or if any files passed as command line arguments do not exist, the example calls the SetError method to redirect error information to a file, calls the OpenStandardError method in the process of reacquiring the standard error stream, and indicates that error information was written to a file.

C#

```
using System;
using System.IO;

public class ViewTextFile
```

```
{
   public static void Main()
   {
      String[] args = Environment.GetCommandLineArgs();
      String errorOutput = "";
      // Make sure that there is at least one command line argument.
      if (args.Length <= 1)
         errorOutput += "You must include a filename on the command
line.\n";

      for (int ctr = 1; ctr <= args.GetUpperBound(0); ctr++)  {
         // Check whether the file exists.
         if (! File.Exists(args[ctr])) {
            errorOutput += String.Format("'{0}' does not exist.\n",
args[ctr]);
         }
         else {
            // Display the contents of the file.
            StreamReader sr = new StreamReader(args[ctr]);
            String contents = sr.ReadToEnd();
            sr.Close();
            Console.WriteLine("*****Contents of file '{0}':\n\n",
                              args[ctr]);
            Console.WriteLine(contents);
            Console.WriteLine("*****\n");
         }
      }

      // Check for error conditions.
      if (! String.IsNullOrEmpty(errorOutput)) {
         // Write error information to a file.
         Console.SetError(new StreamWriter(@".\ViewTextFile.Err.txt"));
         Console.Error.WriteLine(errorOutput);
         Console.Error.Close();
         // Reacquire the standard error stream.
         var standardError = new StreamWriter(Console.OpenStandardError());
         standardError.AutoFlush = true;
         Console.SetError(standardError);
         Console.Error.WriteLine("\nError information written to
ViewTextFile.Err.txt");
      }
   }
}
// If the example is compiled and run with the following command line:
//     ViewTextFile file1.txt file2.txt
// and neither file1.txt nor file2.txt exist, it displays the
// following output:
//     Error information written to ViewTextFile.Err.txt
// and writes the following text to ViewTextFile.Err.txt:
//     'file1.txt' does not exist.
//     'file2.txt' does not exist.
```

Note that the StreamWriter.AutoFlush property is set to `true` before reacquiring the error stream. This ensures that output will be sent to the console immediately rather than buffered.

Remarks

This standard error stream is set to the console by default. It can be set to another stream with the SetError method. After the standard error stream is redirected, it can be reacquired by calling the OpenStandardError method.

In console applications whose informational output is often redirected to a file, the standard error stream available through the Error property can be used to display information to the console even if output is redirected. The following example displays product tables for 10 numbers at a time starting with 1. After every set of 10 numbers, the Error property is used to ask the user whether to display the next set. If the standard output is redirected to a file, the user is still asked whether the routine should generate the next set of products.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int increment = 0;
      bool exitFlag = false;

      while (! exitFlag) {
         if (Console.IsOutputRedirected)
            Console.Error.WriteLine("Generating multiples of numbers from
{0} to {1}",
                                    increment + 1, increment + 10);

         Console.WriteLine("Generating multiples of numbers from {0} to
{1}",
                           increment + 1, increment + 10);
         for (int ctr = increment + 1; ctr <= increment + 10; ctr++) {
            Console.Write("Multiples of {0}: ", ctr);
            for (int ctr2 = 1; ctr2 <= 10; ctr2++)
               Console.Write("{0}{1}", ctr * ctr2, ctr2 == 10 ? "" : ", ");

            Console.WriteLine();
         }
         Console.WriteLine();

         increment += 10;
```

```
            Console.Error.Write("Display multiples of {0} through {1} (y/n)? ",
                                increment + 1, increment + 10);
            Char response = Console.ReadKey(true).KeyChar;
            Console.Error.WriteLine(response);
            if (! Console.IsOutputRedirected)
               Console.CursorTop--;

            if (Char.ToUpperInvariant(response) == 'N')
               exitFlag = true;
         }
      }
   }
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- In
- Out
- IsErrorRedirected
- SetError(TextWriter)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.ForegroundColor Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the foreground color of the console.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static ConsoleColor ForegroundColor { get; set; }
```

Property Value

ConsoleColor

A ConsoleColor that specifies the foreground color of the console; that is, the color of each character that is displayed. The default is gray.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentException

The color specified in a set operation is not a valid member of ConsoleColor.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

The following example checks whether the console's background color is black and, if it is, it changes the background color to red and the foreground color to black.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      if (Console.BackgroundColor == ConsoleColor.Black) {
         Console.BackgroundColor = ConsoleColor.Red;
         Console.ForegroundColor = ConsoleColor.Black;
         Console.Clear();
      }
   }
}
```

The following example saves the values of the ConsoleColor enumeration to an array and stores the current values of the BackgroundColor and ForegroundColor properties to variables. It then changes the foreground color to each color in the ConsoleColor enumeration except to the color that matches the current background, and it changes the background color to each color in the ConsoleColor enumeration except to the color that matches the current foreground. (If the foreground color is the same as the background color, the text isn't visible.) Finally, it calls the ResetColor method to restore the original console colors.

C#

```
using System;

class Example
{
   public static void Main()
   {
      // Get an array with the values of ConsoleColor enumeration members.
      ConsoleColor[] colors = (ConsoleColor[])
ConsoleColor.GetValues(typeof(ConsoleColor));
      // Save the current background and foreground colors.
      ConsoleColor currentBackground = Console.BackgroundColor;
      ConsoleColor currentForeground = Console.ForegroundColor;

      // Display all foreground colors except the one that matches the
background.
      Console.WriteLine("All the foreground colors except {0}, the
background color:",
                        currentBackground);
      foreach (var color in colors) {
         if (color == currentBackground) continue;
```

```
         Console.ForegroundColor = color;
         Console.WriteLine("   The foreground color is {0}.", color);
      }
      Console.WriteLine();
      // Restore the foreground color.
      Console.ForegroundColor = currentForeground;

      // Display each background color except the one that matches the
current foreground color.
      Console.WriteLine("All the background colors except {0}, the
foreground color:",
                        currentForeground);
      foreach (var color in colors) {
         if (color == currentForeground) continue;

         Console.BackgroundColor = color;
         Console.WriteLine("   The background color is {0}.", color);
      }

      // Restore the original console colors.
      Console.ResetColor();
      Console.WriteLine("\nOriginal colors restored...");
   }
}
//The example displays output like the following:
//    All the foreground colors except DarkCyan, the background color:
//       The foreground color is Black.
//       The foreground color is DarkBlue.
//       The foreground color is DarkGreen.
//       The foreground color is DarkRed.
//       The foreground color is DarkMagenta.
//       The foreground color is DarkYellow.
//       The foreground color is Gray.
//       The foreground color is DarkGray.
//       The foreground color is Blue.
//       The foreground color is Green.
//       The foreground color is Cyan.
//       The foreground color is Red.
//       The foreground color is Magenta.
//       The foreground color is Yellow.
//       The foreground color is White.
//
//    All the background colors except White, the foreground color:
//       The background color is Black.
//       The background color is DarkBlue.
//       The background color is DarkGreen.
//       The background color is DarkCyan.
//       The background color is DarkRed.
//       The background color is DarkMagenta.
//       The background color is DarkYellow.
//       The background color is Gray.
//       The background color is DarkGray.
//       The background color is Blue.
//       The background color is Green.
```

```
//       The background color is Cyan.
//       The background color is Red.
//       The background color is Magenta.
//       The background color is Yellow.
//
//    Original colors restored...
```

Remarks

A get operation for a Windows-based application, in which a console does not exist, returns Gray. Unix systems don't provide any general mechanism to fetch the current console colors. Because of that, ForegroundColor returns `(ConsoleColor)-1` until it is set in explicit way (using the setter).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.In Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the standard input stream.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static System.IO.TextReader In { get; }
```

Property Value

TextReader

A TextReader that represents the standard input stream.

Attributes UnsupportedOSPlatformAttribute

Examples

The following sample illustrates the use of the `In` property.

C#

```
using System;
using System.IO;

class InTest {
    public static void Main() {

        TextReader tIn = Console.In;
        TextWriter tOut = Console.Out;

        tOut.WriteLine("Hola Mundo!");
        tOut.Write("What is your name: ");
        String name = tIn.ReadLine();
```

```
        tOut.WriteLine("Buenos Dias, {0}!", name);
    }
}
```

Remarks

This property is set to the standard input stream by default. This property can be set to another stream with the SetIn method.

Read operations on the standard input stream execute synchronously. That is, they block until the specified read operation has completed. This is true even if an asynchronous method, such as ReadLineAsync, is called on the TextReader object returned by the In property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- OpenStandardInput()
- IsInputRedirected
- Error
- Out
- SetIn(TextReader)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Console.InputEncoding Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the encoding the console uses to read input.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static System.Text.Encoding InputEncoding { get; set; }
```

Property Value

Encoding

The encoding used to read console input.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentNullException

The property value in a set operation is `null`.

IOException

An error occurred during the execution of this operation.

SecurityException

Your application does not have permission to perform this operation.

Remarks

The console uses the input encoding to translate keyboard input into a corresponding character. The input encoding incorporates a code page that maps 256 keyboard character codes to individual characters. Different code pages include different special characters, typically customized for a language or a group of languages.

Starting with the .NET Framework 4, a property get operation may return a cached value instead of the console's current input encoding. This can occur if the value of the InputEncoding property is modified by some means other than an assignment to the InputEncoding property, such as calling the Windows `SetConsoleCP` function or using the `chcp` command from a PowerShell script.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Encoding

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.IsErrorRedirected Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value that indicates whether the error output stream has been redirected from the standard error stream.

```
C#
public static bool IsErrorRedirected { get; }
```

Property Value

Boolean

`true` if error output is redirected; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Error
- SetError(TextWriter)
- OpenStandardError()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.IsInputRedirected Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value that indicates whether input has been redirected from the standard input stream.

```
C#
public static bool IsInputRedirected { get; }
```

Property Value

Boolean

`true` if input is redirected; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- In
- SetIn(TextReader)
- OpenStandardInput()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.IsOutputRedirected Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value that indicates whether output has been redirected from the standard output stream.

```
C#
public static bool IsOutputRedirected { get; }
```

Property Value

Boolean

`true` if output is redirected; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Out
- SetOut(TextWriter)
- OpenStandardError()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.KeyAvailable Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value indicating whether a key press is available in the input stream.

C#

```
public static bool KeyAvailable { get; }
```

Property Value

Boolean

`true` if a key press is available; otherwise, `false`.

Exceptions

IOException

An I/O error occurred.

InvalidOperationException

Standard input is redirected to a file instead of the keyboard.

Examples

The following example demonstrates how to use the KeyAvailable property to create a loop that runs until a key is pressed.

C#

```
using System;
using System.Threading;

class Sample
{
    public static void Main()
    {
```

```
        ConsoleKeyInfo cki;

        do {
            Console.WriteLine("\nPress a key to display; press the 'x' key to
quit.");

            // Your code could perform some useful task in the following
loop. However,
            // for the sake of this example we'll merely pause for a quarter
second.

            while (Console.KeyAvailable == false)
                Thread.Sleep(250); // Loop until input is entered.

            cki = Console.ReadKey(true);
            Console.WriteLine("You pressed the '{0}' key.", cki.Key);
        } while(cki.Key != ConsoleKey.X);
    }
}
/*
This example produces results similar to the following:

Press a key to display; press the 'x' key to quit.
You pressed the 'H' key.

Press a key to display; press the 'x' key to quit.
You pressed the 'E' key.

Press a key to display; press the 'x' key to quit.
You pressed the 'PageUp' key.

Press a key to display; press the 'x' key to quit.
You pressed the 'DownArrow' key.

Press a key to display; press the 'x' key to quit.
You pressed the 'X' key.
*/
```

Remarks

The property value is returned immediately; that is, the KeyAvailable property does not block input until a key press is available.

Use the KeyAvailable property in conjunction with only the ReadKey method, not the Read or ReadLine methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.LargestWindowHeight Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the largest possible number of console window rows, based on the current font and screen resolution.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int LargestWindowHeight { get; }
```

Property Value

Int32

The height of the largest possible console window measured in rows.

Attributes UnsupportedOSPlatformAttribute

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.LargestWindowWidth Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the largest possible number of console window columns, based on the current font and screen resolution.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int LargestWindowWidth { get; }
```

Property Value

Int32

The width of the largest possible console window measured in columns.

Attributes UnsupportedOSPlatformAttribute

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.NumberLock Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value indicating whether the NUM LOCK keyboard toggle is turned on or turned off.

```C#
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static bool NumberLock { get; }
```

Property Value

Boolean

`true` if NUM LOCK is turned on; `false` if NUM LOCK is turned off.

Attributes SupportedOSPlatformAttribute

Exceptions

PlatformNotSupportedException

The get operation is invoked on an operating system other than Windows.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.Out Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the standard output stream.

C#

```
public static System.IO.TextWriter Out { get; }
```

Property Value

TextWriter

A TextWriter that represents the standard output stream.

Examples

The following example uses the Out property to display an array containing the names of files in the application's current directory to the standard output device. It then sets the standard output to a file named Files.txt and lists the array elements to the file. Finally, it sets the output to the standard output stream and again displays the array elements to the standard output device.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      // Get all files in the current directory.
      string[] files = Directory.GetFiles(".");
      Array.Sort(files);

      // Display the files to the current output source to the console.
      Console.Out.WriteLine("First display of filenames to the console:");
```

```
        Array.ForEach(files, s => Console.Out.WriteLine(s));
        Console.Out.WriteLine();

        // Redirect output to a file named Files.txt and write file list.
        StreamWriter sw = new StreamWriter(@".\Files.txt");
        sw.AutoFlush = true;
        Console.SetOut(sw);
        Console.Out.WriteLine("Display filenames to a file:");
        Array.ForEach(files, s => Console.Out.WriteLine(s));
        Console.Out.WriteLine();

        // Close previous output stream and redirect output to standard
output.
        Console.Out.Close();
        sw = new StreamWriter(Console.OpenStandardOutput());
        sw.AutoFlush = true;
        Console.SetOut(sw);

        // Display the files to the current output source to the console.
        Console.Out.WriteLine("Second display of filenames to the console:");
        Array.ForEach(files, s => Console.Out.WriteLine(s));
    }
}
```

Remarks

This property is set to the standard output stream by default. This property can be set to another stream with the SetOut method.

Note that calls to `Console.Out.WriteLine` methods are equivalent to calls to the corresponding WriteLine methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Error
- In

- SetOut(TextWriter)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.OutputEncoding Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the encoding the console uses to write output.

C#

```
public static System.Text.Encoding OutputEncoding { get;
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] set; }
```

Property Value

Encoding

The encoding used to write console output.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentNullException

The property value in a set operation is `null`.

IOException

An error occurred during the execution of this operation.

SecurityException

Your application does not have permission to perform this operation.

Remarks

The console uses the output encoding to translate characters written by an application into corresponding console display characters. The default code page that the console

uses is determined by the system locale.

Starting with the .NET Framework 4, a property get operation may return a cached value instead of the console's current output encoding. This can occur if the value of the OutputEncoding property is modified by some means other than an assignment to the OutputEncoding property, such as calling the Windows `SetConsoleOutputCP` function.

Notes to Callers

Of the Unicode encodings, the Console class supports UTF-8 encoding with the UTF8Encoding class and, starting with the .NET Framework 4.5, it supports UTF-16 encoding with the UnicodeEncoding class. UTF-32 encoding with the UTF32Encoding class is not supported. Attempting to set the output encoding to UTF-32 throws an IOException.

Note that successfully displaying Unicode characters to the console requires the following:

- The console must use a TrueType font, such as Lucida Console or Consolas, to display characters.
- A font used by the console must define the particular glyph or glyphs to be displayed. The console can take advantage of font linking to display glyphs from linked fonts if the base font does not contain a definition for that glyph.

For more information about support for Unicode encoding by the console, see the "Unicode Support for the Console" section in the Console class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.Title Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the title to display in the console title bar.

C#

```
public static string Title {
[System.Runtime.Versioning.SupportedOSPlatform("windows")] get;
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")] set; }
```

Property Value

String

The string to be displayed in the title bar of the console. The maximum length of the title string is 24500 characters.

Attributes SupportedOSPlatformAttribute, UnsupportedOSPlatformAttribute

Exceptions

InvalidOperationException

In a get operation, the retrieved title is longer than 24500 characters.

ArgumentOutOfRangeException

In a set operation, the specified title is longer than 24500 characters.

ArgumentNullException

In a set operation, the specified title is `null`.

IOException

An I/O error occurred.

PlatformNotSupportedException

The get operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the Title property. The example displays the current title of the operating system window, waits for a key press, then displays a new title.

C#

```
// This example demonstrates the Console.Title property.
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine("The current console title is: \"{0}\"",
                      Console.Title);
    Console.WriteLine("  (Press any key to change the console title.)");
    Console.ReadKey(true);
    Console.Title = "The title has changed!";
    Console.WriteLine("Note that the new console title is \"{0}\"\n" +
                      "  (Press any key to quit.)", Console.Title);
    Console.ReadKey(true);
    }
}
/*
This example produces the following results:

>myTitle
The current console title is: "Command Prompt - myTitle"
  (Press any key to change the console title.)
Note that the new console title is "The title has changed!"
  (Press any key to quit.)

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.TreatControlCAsInput Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets a value indicating whether the combination of the Control modifier key and C console key (Ctrl+C) is treated as ordinary input or as an interruption that is handled by the operating system.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static bool TreatControlCAsInput { get; set; }
```

Property Value

Boolean

`true` if Ctrl+C is treated as ordinary input; otherwise, `false`.

Attributes UnsupportedOSPlatformAttribute

Exceptions

IOException

Unable to get or set the input mode of the console input buffer.

Examples

The following example demonstrates the TreatControlCAsInput property.

C#

```
using System;

class Example
```

```
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT, and a
console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
         if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
         if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
         if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
         Console.WriteLine(cki.Key.ToString());
       } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Remarks

If the value of the TreatControlCAsInput property is `false` and Ctrl+C is pressed, the pressed keys are not stored in the input buffer and the operating system terminates the currently executing process. This is the default value.

> ⊗ **Caution**
>
> Use this property judiciously because setting it to `true` has such a dramatic effect. Most users expect Ctrl+C to terminate a console application. If you disable the

effect of Ctrl+C, the user must remember to use Ctrl+Break to terminate the application, which is a less familiar key combination.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Control
- C
- CancelKeyPress

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.WindowHeight Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the height of the console window area.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int WindowHeight { get; set; }
```

Property Value

Int32

The height of the console window measured in rows.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value of the WindowWidth property or the value of the WindowHeight property is less than or equal to 0.

-or-

The value of the WindowHeight property plus the value of the WindowTop property is greater than or equal to Int16.MaxValue.

-or-

The value of the WindowWidth property or the value of the WindowHeight property is greater than the largest possible window width or height for the current screen resolution and console font.

IOException

Error reading or writing information.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the SetWindowSize method, and the WindowWidth and WindowHeight properties. You must run the example to see the full effect of changing the console window size.

The example reports the dimensions of a console window set to 85 columns and 43 rows, then waits for a key press. When any key is pressed, the dimensions of the console window are halved, the new dimensions are reported, and the example waits for another key press. Finally, when any key is pressed the console window is restored to its original dimensions and the example terminates.

C#

```
// This example demonstrates the Console.SetWindowSize method,
//                            the Console.WindowWidth property,
//                        and the Console.WindowHeight property.
using System;

class Sample
{
    public static void Main()
    {
    int origWidth, width;
    int origHeight, height;
    string m1 = "The current window width is {0}, and the " +
                "current window height is {1}.";
    string m2 = "The new window width is {0}, and the new " +
                "window height is {1}.";
    string m4 = "  (Press any key to continue...)";
//
// Step 1: Get the current window dimensions.
//
    origWidth  = Console.WindowWidth;
    origHeight = Console.WindowHeight;
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 2: Cut the window to 1/4 its original size.
//
```

```
    width  = origWidth/2;
    height = origHeight/2;
    Console.SetWindowSize(width, height);
    Console.WriteLine(m2, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 3: Restore the window to its original size.
//
    Console.SetWindowSize(origWidth, origHeight);
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    }
}
/*
This example produces the following results:

The current window width is 85, and the current window height is 43.
  (Press any key to continue...)
The new window width is 42, and the new window height is 21.
  (Press any key to continue...)
The current window width is 85, and the current window height is 43.

*/
```

Remarks

Attempting to set the value of the WindowHeight property when output is redirected throws either an ArgumentOutOfRangeException or an IOException exception. To prevent an exception, you can set the value of this property only if the IsOutputRedirected property returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.WindowLeft Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the leftmost position of the console window area relative to the screen buffer.

C#

```
public static int WindowLeft { get;
[System.Runtime.Versioning.SupportedOSPlatform("windows")] set; }
```

Property Value

Int32

The leftmost console window position measured in columns.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

In a set operation, the value to be assigned is less than zero.

-or-

As a result of the assignment, WindowLeft plus WindowWidth would exceed BufferWidth.

IOException

Error reading or writing information.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

The following example opens an 80-column console window and defines a buffer area that is 120 columns wide. It displays information on window and buffer size, and then waits for the user to press either the LEFT ARROW key or the RIGHT ARROW key. In the former case, it decrements the value of the WindowLeft property by one if the result is a legal value. In the latter case, it increases the value of the WindowLeft property by one if the result would be legal. Note that the example does not have to handle an ArgumentOutOfRangeException, because it checks that the value to be assigned to the WindowLeft property is not negative and does not cause the sum of the WindowLeft and WindowWidth properties to exceed the BufferWidth property value.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      ConsoleKeyInfo key;
      bool moved = false;

      Console.BufferWidth += 4;
      Console.Clear();

      ShowConsoleStatistics();
      do
      {
         key = Console.ReadKey(true);
         if (key.Key == ConsoleKey.LeftArrow)
         {
            int pos = Console.WindowLeft - 1;
            if (pos >= 0 && pos + Console.WindowWidth <=
Console.BufferWidth)
            {
               Console.WindowLeft = pos;
               moved = true;
            }
         }
         else if (key.Key == ConsoleKey.RightArrow)
         {
            int pos = Console.WindowLeft + 1;
            if (pos + Console.WindowWidth <= Console.BufferWidth)
            {
               Console.WindowLeft = pos;
               moved = true;
            }
         }
         if (moved)
```

```
            {
               ShowConsoleStatistics();
               moved = false;
            }
            Console.WriteLine();
         } while (true);
      }

      private static void ShowConsoleStatistics()
      {
         Console.WriteLine("Console statistics:");
         Console.WriteLine("   Buffer: {0} x {1}", Console.BufferHeight,
Console.BufferWidth);
         Console.WriteLine("   Window: {0} x {1}", Console.WindowHeight,
Console.WindowWidth);
         Console.WriteLine("   Window starts at {0}.", Console.WindowLeft);
         Console.WriteLine("Press <- or -> to move window, Ctrl+C to exit.");
      }
   }
```

Remarks

The console represents a rectangular window into a larger rectangular buffer area. Both the window and the buffer are measured vertically by their number of rows and horizontally by their number of columns. The dimensions of the buffer area are defined by the BufferHeight and BufferWidth properties. The dimensions of the console area are defined by the WindowHeight and WindowWidth properties. The WindowLeft property determines which column of the buffer area is displayed in the first column of the console window. The value of the WindowLeft property can range from 0 to BufferWidth - WindowWidth. Attempting to set it to a value outside that range throws an ArgumentOutOfRangeException.

When a console window first opens, the default value of the WindowLeft property is zero, which indicates that the first column shown by the console corresponds to the first column (the column at position zero) in the buffer area. The default width of both the console window and the buffer area is 80 columns. This means that the WindowLeft property can be modified only if the console window is made narrower or the buffer area is made wider.

Note that if the width of the buffer area exceeds the width of the console window, the value of the WindowLeft property is automatically adjusted when the user uses the horizontal scroll bar to define the window's relationship to the buffer area.

Attempting to set the value of the WindowLeft property when output is redirected throws an IOException exception. To prevent the exception, you can set the value of this

property only if the IsOutputRedirected property returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.WindowTop Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the top position of the console window area relative to the screen buffer.

C#

```
public static int WindowTop { get;
[System.Runtime.Versioning.SupportedOSPlatform("windows")] set; }
```

Property Value

Int32

The uppermost console window position measured in rows.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

In a set operation, the value to be assigned is less than zero.

-or-

As a result of the assignment, WindowTop plus WindowHeight would exceed BufferHeight.

IOException

Error reading or writing information.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

The following example demonstrates the WindowLeft, WindowTop, WindowWidth, WindowHeight, BufferWidth, BufferHeight, and CursorVisible properties; and the SetWindowPosition, SetBufferSize, and ReadKey methods. The example draws a grid pattern in the screen buffer based on the screen buffer width. Then the example moves the console window in response to which of the UP ARROW, DOWN ARROW, LEFT ARROW, or RIGHT ARROW console keys is pressed. The grid pattern helps you see the movement of the console window relative to the screen buffer.

C#

```
// This example demonstrates the Console.WindowLeft and
//                               Console.WindowTop properties.
using System;
using System.Text;
using System.IO;
//
class Sample
{
    public static int saveBufferWidth;
    public static int saveBufferHeight;
    public static int saveWindowHeight;
    public static int saveWindowWidth;
    public static bool saveCursorVisible;
//
    public static void Main()
    {
    string m1 = "1) Press the cursor keys to move the console window.\n" +
                "2) Press any key to begin. When you're finished...\n" +
                "3) Press the Escape key to quit.";
    string g1 = "+----";
    string g2 = "|    ";
    string grid1;
    string grid2;
    StringBuilder sbG1 = new StringBuilder();
    StringBuilder sbG2 = new StringBuilder();
    ConsoleKeyInfo cki;
    int y;
//
    try
    {
    saveBufferWidth  = Console.BufferWidth;
    saveBufferHeight = Console.BufferHeight;
    saveWindowHeight = Console.WindowHeight;
    saveWindowWidth  = Console.WindowWidth;
    saveCursorVisible = Console.CursorVisible;
//
    Console.Clear();
    Console.WriteLine(m1);
    Console.ReadKey(true);

// Set the smallest possible window size before setting the buffer size.
    Console.SetWindowSize(1, 1);
```

```
    Console.SetBufferSize(80, 80);
    Console.SetWindowSize(40, 20);

// Create grid lines to fit the buffer. (The buffer width is 80, but
// this same technique could be used with an arbitrary buffer width.)
    for (y = 0; y < Console.BufferWidth/g1.Length; y++)
        {
        sbG1.Append(g1);
        sbG2.Append(g2);
        }
    sbG1.Append(g1, 0, Console.BufferWidth%g1.Length);
    sbG2.Append(g2, 0, Console.BufferWidth%g2.Length);
    grid1 = sbG1.ToString();
    grid2 = sbG2.ToString();

    Console.CursorVisible = false;
    Console.Clear();
    for (y = 0; y < Console.BufferHeight-1; y++)
        {
        if (y%3 == 0)
            Console.Write(grid1);
        else
            Console.Write(grid2);
        }

    Console.SetWindowPosition(0, 0);
    do
        {
        cki = Console.ReadKey(true);
        switch (cki.Key)
            {
            case ConsoleKey.LeftArrow:
                if (Console.WindowLeft > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft-1, Console.WindowTop);
                break;
            case ConsoleKey.UpArrow:
                if (Console.WindowTop > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop-1);
                break;
            case ConsoleKey.RightArrow:
                if (Console.WindowLeft < (Console.BufferWidth-
Console.WindowWidth))
                    Console.SetWindowPosition(
                            Console.WindowLeft+1, Console.WindowTop);
                break;
            case ConsoleKey.DownArrow:
                if (Console.WindowTop < (Console.BufferHeight-
Console.WindowHeight))
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop+1);
                break;
            }
        }
```

```
    while (cki.Key != ConsoleKey.Escape);  // end do-while
    } // end try
    catch (IOException e)
        {
        Console.WriteLine(e.Message);
        }
    finally
        {
        Console.Clear();
        Console.SetWindowSize(1, 1);
        Console.SetBufferSize(saveBufferWidth, saveBufferHeight);
        Console.SetWindowSize(saveWindowWidth, saveWindowHeight);
        Console.CursorVisible = saveCursorVisible;
        }
    } // end Main
} // end Sample
/*
This example produces results similar to the following:

1) Press the cursor keys to move the console window.
2) Press any key to begin. When you're finished...
3) Press the Escape key to quit.

...

+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-

*/
```

Remarks

The console represents a rectangular window into a larger rectangular buffer area. Both the window and the buffer are measured vertically by their number of rows and horizontally by their number of columns. The dimensions of the buffer area are defined by the BufferHeight and BufferWidth properties. The dimensions of the console area are defined by the WindowHeight and WindowWidth properties. The WindowTop property determines which row of the buffer area is displayed in the first column of the console window. The value of the WindowTop property can range from 0 to BufferHeight - WindowHeight. Attempting to set it to a value outside that range throws an ArgumentOutOfRangeException.

Attempting to set the value of the WindowTop property when output is redirected throws an IOException exception. To prevent the exception, you can set the value of this

property only if the IsOutputRedirected property returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.WindowWidth Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets the width of the console window.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static int WindowWidth { get; set; }
```

Property Value

Int32

The width of the console window measured in columns.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

The value of the WindowWidth property or the value of the WindowHeight property is less than or equal to 0.

-or-

The value of the WindowHeight property plus the value of the WindowTop property is greater than or equal to Int16.MaxValue.

-or-

The value of the WindowWidth property or the value of the WindowHeight property is greater than the largest possible window width or height for the current screen resolution and console font.

IOException

Error reading or writing information.

PlatformNotSupportedException

The set operation is invoked on an operating system other than Windows.

Examples

This example demonstrates the SetWindowSize method, and the WindowWidth and WindowHeight properties. You must run the example to see the full effect of changing the console window size.

The example reports the dimensions of a console window set to 85 columns and 43 rows, then waits for a key press. When any key is pressed, the dimensions of the console window are halved, the new dimensions are reported, and the example waits for another key press. Finally, when any key is pressed the console window is restored to its original dimensions and the example terminates.

C#

```
// This example demonstrates the Console.SetWindowSize method,
//                            the Console.WindowWidth property,
//                        and the Console.WindowHeight property.
using System;

class Sample
{
    public static void Main()
    {
    int origWidth, width;
    int origHeight, height;
    string m1 = "The current window width is {0}, and the " +
                "current window height is {1}.";
    string m2 = "The new window width is {0}, and the new " +
                "window height is {1}.";
    string m4 = "  (Press any key to continue...)";
//
// Step 1: Get the current window dimensions.
//
    origWidth  = Console.WindowWidth;
    origHeight = Console.WindowHeight;
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 2: Cut the window to 1/4 its original size.
//
```

```
    width  = origWidth/2;
    height = origHeight/2;
    Console.SetWindowSize(width, height);
    Console.WriteLine(m2, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 3: Restore the window to its original size.
//
    Console.SetWindowSize(origWidth, origHeight);
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    }
}
/*
This example produces the following results:

The current window width is 85, and the current window height is 43.
  (Press any key to continue...)
The new window width is 42, and the new window height is 21.
  (Press any key to continue...)
The current window width is 85, and the current window height is 43.

*/
```

Remarks

Attempting to set the value of the WindowWidth property when output is redirected throws either an ArgumentOutOfRangeException or an IOException exception. To prevent an exception, you can set the value of this property only if the IsOutputRedirected property returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.Beep Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Plays the sound of a beep through the console speaker.

Overloads

Expand table

Beep()	Plays the sound of a beep through the console speaker.
Beep(Int32, Int32)	Plays the sound of a beep of a specified frequency and duration through the console speaker.

Beep()

Plays the sound of a beep through the console speaker.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void Beep ();
```

Attributes UnsupportedOSPlatformAttribute

Exceptions

HostProtectionException

This method was executed on a server, such as SQL Server, that does not permit access to a user interface.

Examples

The following example demonstrates the Beep method. The example accepts a number from 1 through 9 as a command line argument, and plays the beep that number of times.

C#

```
// This example demonstrates the Console.Beep() method.
using System;

class Sample
{
    public static void Main(String[] args)
    {
    int x = 0;
//
    if ((args.Length == 1) &&
        (Int32.TryParse(args[0], out x) == true) &&
        ((x >= 1) && (x <= 9)))
        {
        for (int i = 1; i <= x; i++)
            {
            Console.WriteLine("Beep number {0}.", i);
            Console.Beep();
            }
        }
    else
        {
            Console.WriteLine("Usage: Enter the number of times (between
1 and 9) to beep.");
        }
    }
}
/*
This example produces the following results:

>beep
Usage: Enter the number of times (between 1 and 9) to beep

>beep 9
Beep number 1.
Beep number 2.
Beep number 3.
Beep number 4.
Beep number 5.
Beep number 6.
Beep number 7.
Beep number 8.
Beep number 9.

*/
```

Remarks

By default, the beep plays at a frequency of 800 hertz for a duration of 200 milliseconds.

Beep wraps a call to the Windows Beep function. Whether Beep produces a sound on versions of Windows before Windows 7 depends on the presence of a 8254 programmable interval timer chip. Starting with Windows 7, it depends on the default sound device.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Beep(Int32, Int32)

Plays the sound of a beep of a specified frequency and duration through the console speaker.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static void Beep (int frequency, int duration);
```

Parameters

`frequency` Int32

The frequency of the beep, ranging from 37 to 32767 hertz.

`duration` Int32

The duration of the beep measured in milliseconds.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`frequency` is less than 37 or more than 32767 hertz.

-or-

`duration` is less than or equal to zero.

HostProtectionException

This method was executed on a server, such as SQL Server, that does not permit access to the console.

PlatformNotSupportedException

The current operating system is not Windows.

Examples

This example demonstrates the Beep method by playing the first few notes of a song through the console speaker.

C#

```
// This example demonstrates the Console.Beep(Int32, Int32) method
using System;
using System.Threading;

class Sample
{
    public static void Main()
    {
// Declare the first few notes of the song, "Mary Had A Little Lamb".
    Note[] Mary =
        {
        new Note(Tone.B, Duration.QUARTER),
        new Note(Tone.A, Duration.QUARTER),
        new Note(Tone.GbelowC, Duration.QUARTER),
        new Note(Tone.A, Duration.QUARTER),
        new Note(Tone.B, Duration.QUARTER),
        new Note(Tone.B, Duration.QUARTER),
        new Note(Tone.B, Duration.HALF),
        new Note(Tone.A, Duration.QUARTER),
        new Note(Tone.A, Duration.QUARTER),
        new Note(Tone.A, Duration.HALF),
        new Note(Tone.B, Duration.QUARTER),
```

```
        new Note(Tone.D, Duration.QUARTER),
        new Note(Tone.D, Duration.HALF)
        };
// Play the song
    Play(Mary);
    }

// Play the notes in a song.
    protected static void Play(Note[] tune)
    {
    foreach (Note n in tune)
        {
        if (n.NoteTone == Tone.REST)
            Thread.Sleep((int)n.NoteDuration);
        else
            Console.Beep((int)n.NoteTone, (int)n.NoteDuration);
        }
    }

// Define the frequencies of notes in an octave, as well as
// silence (rest).
    protected enum Tone
    {
    REST   = 0,
    GbelowC = 196,
    A      = 220,
    Asharp = 233,
    B      = 247,
    C      = 262,
    Csharp = 277,
    D      = 294,
    Dsharp = 311,
    E      = 330,
    F      = 349,
    Fsharp = 370,
    G      = 392,
    Gsharp = 415,
    }

// Define the duration of a note in units of milliseconds.
    protected enum Duration
    {
    WHOLE     = 1600,
    HALF      = WHOLE/2,
    QUARTER   = HALF/2,
    EIGHTH    = QUARTER/2,
    SIXTEENTH = EIGHTH/2,
    }

// Define a note as a frequency (tone) and the amount of
// time (duration) the note plays.
    protected struct Note
    {
    Tone     toneVal;
    Duration durVal;
```

```
// Define a constructor to create a specific note.
    public Note(Tone frequency, Duration time)
        {
        toneVal = frequency;
        durVal  = time;
        }

// Define properties to return the note's tone and duration.
    public Tone NoteTone { get{ return toneVal; } }
    public Duration NoteDuration { get{ return durVal; } }
    }
}
/*
This example produces the following results:

This example plays the first few notes of "Mary Had A Little Lamb"
through the console speaker.
*/
```

Remarks

Beep wraps a call to the Windows Beep function. Whether Beep produces a sound on versions of Windows before Windows 7 depends on the presence of a 8254 programmable interval timer chip. Starting with Windows 7, it depends on the default sound device.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Console.Clear Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Clears the console buffer and corresponding console window of display information.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void Clear ();
```

Attributes UnsupportedOSPlatformAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example uses the Clear method to clear the console before it executes a loop, prompts the user to select a foreground and background color and to enter a string to display. If the user chooses not to exit the program, the console's original foreground and background colors are restored and the Clear method is called again before re-executing the loop.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Save colors so they can be restored when use finishes input.
      ConsoleColor dftForeColor = Console.ForegroundColor;
```

```
      ConsoleColor dftBackColor = Console.BackgroundColor;
      bool continueFlag = true;
      Console.Clear();

      do {
         ConsoleColor newForeColor = ConsoleColor.White;
         ConsoleColor newBackColor = ConsoleColor.Black;

         Char foreColorSelection = GetKeyPress("Select Text Color (B for
Blue, R for Red, Y for Yellow): ",
                                               new Char[] { 'B', 'R', 'Y' }
);
         switch (foreColorSelection) {
            case 'B':
            case 'b':
               newForeColor = ConsoleColor.DarkBlue;
               break;
            case 'R':
            case 'r':
               newForeColor = ConsoleColor.DarkRed;
               break;
            case 'Y':
            case 'y':
               newForeColor = ConsoleColor.DarkYellow;
               break;
         }
         Char backColorSelection = GetKeyPress("Select Background Color (W
for White, G for Green, M for Magenta): ",
                                               new Char[] { 'W', 'G', 'M' });
         switch (backColorSelection) {
            case 'W':
            case 'w':
               newBackColor = ConsoleColor.White;
               break;
            case 'G':
            case 'g':
               newBackColor = ConsoleColor.Green;
               break;
            case 'M':
            case 'm':
               newBackColor = ConsoleColor.Magenta;
               break;
         }

         Console.WriteLine();
         Console.Write("Enter a message to display: ");
         String textToDisplay = Console.ReadLine();
         Console.WriteLine();
         Console.ForegroundColor = newForeColor;
         Console.BackgroundColor = newBackColor;
         Console.WriteLine(textToDisplay);
         Console.WriteLine();
         if (Char.ToUpper(GetKeyPress("Display another message (Y/N): ", new
Char[] { 'Y', 'N' } )) == 'N')
            continueFlag = false;
```

```
         // Restore the default settings and clear the screen.
         Console.ForegroundColor = dftForeColor;
         Console.BackgroundColor = dftBackColor;
         Console.Clear();
      } while (continueFlag);
   }

   private static Char GetKeyPress(String msg, Char[] validChars)
   {
      ConsoleKeyInfo keyPressed;
      bool valid = false;

      Console.WriteLine();
      do {
         Console.Write(msg);
         keyPressed = Console.ReadKey();
         Console.WriteLine();
         if (Array.Exists(validChars, ch =>
ch.Equals(Char.ToUpper(keyPressed.KeyChar))))
            valid = true;
      } while (! valid);
      return keyPressed.KeyChar;
   }
}
```

The example relies on a `GetKeyPress` method to validate the user's selection of a foreground and background color.

This example demonstrates the CursorLeft and CursorTop properties, and the SetCursorPosition and Clear methods. The example positions the cursor, which determines where the next write will occur, to draw a 5 character by 5 character rectangle using a combination of "+", "|", and "-" strings. Note that the rectangle could be drawn with fewer steps using a combination of other strings.

C#

```
// This example demonstrates the
//     Console.CursorLeft and
//     Console.CursorTop properties, and the
//     Console.SetCursorPosition and
//     Console.Clear methods.
using System;

class Sample
{
    protected static int origRow;
    protected static int origCol;

    protected static void WriteAt(string s, int x, int y)
    {
```

```
    try
        {
        Console.SetCursorPosition(origCol+x, origRow+y);
        Console.Write(s);
        }
    catch (ArgumentOutOfRangeException e)
        {
        Console.Clear();
        Console.WriteLine(e.Message);
        }
    }

    public static void Main()
    {
// Clear the screen, then save the top and left coordinates.
    Console.Clear();
    origRow = Console.CursorTop;
    origCol = Console.CursorLeft;

// Draw the left side of a 5x5 rectangle, from top to bottom.
    WriteAt("+", 0, 0);
    WriteAt("|", 0, 1);
    WriteAt("|", 0, 2);
    WriteAt("|", 0, 3);
    WriteAt("+", 0, 4);

// Draw the bottom side, from left to right.
    WriteAt("-", 1, 4); // shortcut: WriteAt("---", 1, 4)
    WriteAt("-", 2, 4); // ...
    WriteAt("-", 3, 4); // ...
    WriteAt("+", 4, 4);

// Draw the right side, from bottom to top.
    WriteAt("|", 4, 3);
    WriteAt("|", 4, 2);
    WriteAt("|", 4, 1);
    WriteAt("+", 4, 0);

// Draw the top side, from right to left.
    WriteAt("-", 3, 0); // shortcut: WriteAt("---", 1, 0)
    WriteAt("-", 2, 0); // ...
    WriteAt("-", 1, 0); // ...
//
    WriteAt("All done!", 0, 6);
    Console.WriteLine();
    }
}
/*
This example produces the following results:

+---+
|   |
|   |
|   |
+---+
```

```
All done!

*/
```

Remarks

Using the Clear method is equivalent invoking the MS-DOS `cls` command in the command prompt window. When the Clear method is called, the cursor automatically scrolls to the top-left corner of the window and the contents of the screen buffer are set to blanks using the current foreground background colors.

> **Note**
>
> Attempting to call the **Clear** method when a console application's output is redirected to a file throws a **IOException**. To prevent this, always wrap a call to the **Clear** method in a `try`...`catch` block.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.GetCursorPosition Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the position of the cursor.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static (int Left, int Top) GetCursorPosition ();
```

Returns

ValueTuple<Int32,Int32>

The column and row position of the cursor.

Attributes UnsupportedOSPlatformAttribute

Remarks

Columns are numbered from left to right starting at 0. Rows are numbered from top to bottom starting at 0.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.MoveBufferArea Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Copies a specified source area of the screen buffer to a specified destination area.

Overloads

Expand table

MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32)	Copies a specified source area of the screen buffer to a specified destination area.
MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32, Char, ConsoleColor, ConsoleColor)	Copies a specified source area of the screen buffer to a specified destination area.

MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32)

Copies a specified source area of the screen buffer to a specified destination area.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static void MoveBufferArea (int sourceLeft, int sourceTop, int
sourceWidth, int sourceHeight, int targetLeft, int targetTop);
```

Parameters

`sourceLeft` Int32

The leftmost column of the source area.

`sourceTop` Int32

The topmost row of the source area.

`sourceWidth` Int32

The number of columns in the source area.

`sourceHeight` Int32

The number of rows in the source area.

`targetLeft` Int32

The leftmost column of the destination area.

`targetTop` Int32

The topmost row of the destination area.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

One or more of the parameters is less than zero.

-or-

`sourceLeft` or `targetLeft` is greater than or equal to BufferWidth.

-or-

`sourceTop` or `targetTop` is greater than or equal to BufferHeight.

-or-

`sourceTop` + `sourceHeight` is greater than or equal to BufferHeight.

-or-

`sourceLeft` + `sourceWidth` is greater than or equal to BufferWidth.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The current operating system is not Windows.

Remarks

If the destination and source parameters specify a position located outside the boundaries of the current screen buffer, only the portion of the source area that fits within the destination area is copied. That is, the source area is clipped to fit the current screen buffer.

The MoveBufferArea method copies the source area to the destination area. If the destination area does not intersect the source area, the source area is filled with blanks using the current foreground and background colors. Otherwise, the intersected portion of the source area is not filled.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32, Char, ConsoleColor, ConsoleColor)

Copies a specified source area of the screen buffer to a specified destination area.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static void MoveBufferArea (int sourceLeft, int sourceTop, int
sourceWidth, int sourceHeight, int targetLeft, int targetTop, char
sourceChar, ConsoleColor sourceForeColor, ConsoleColor sourceBackColor);
```

Parameters

`sourceLeft` Int32

The leftmost column of the source area.

`sourceTop` Int32

The topmost row of the source area.

`sourceWidth` Int32

The number of columns in the source area.

`sourceHeight` Int32

The number of rows in the source area.

`targetLeft` Int32

The leftmost column of the destination area.

`targetTop` Int32

The topmost row of the destination area.

`sourceChar` Char

The character used to fill the source area.

`sourceForeColor` ConsoleColor

The foreground color used to fill the source area.

`sourceBackColor` ConsoleColor

The background color used to fill the source area.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

One or more of the parameters is less than zero.

-or-

`sourceLeft` or `targetLeft` is greater than or equal to BufferWidth.

-or-

`sourceTop` or `targetTop` is greater than or equal to BufferHeight.

-or-

`sourceTop` + `sourceHeight` is greater than or equal to BufferHeight.

-or-

`sourceLeft` + `sourceWidth` is greater than or equal to BufferWidth.

ArgumentException

One or both of the color parameters is not a member of the ConsoleColor enumeration.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The current operating system is not Windows.

Remarks

If the destination and source parameters specify a position located beyond the boundaries of the current screen buffer, only the portion of the source area that fits within the destination area is copied. That is, the source area is clipped to fit the current screen buffer.

The MoveBufferArea method copies the source area to the destination area. If the destination area does not intersect the source area, the source area is filled with the character specified by `sourceChar`, using the colors specified by `sourceForeColor` and `sourceBackColor`. Otherwise, the intersected portion of the source area is not filled.

The MoveBufferArea method performs no operation if `sourceWidth` or `sourceHeight` is zero.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.OpenStandardError Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Acquires the standard error stream.

Overloads

Expand table

OpenStandardError()	Acquires the standard error stream.
OpenStandardError(Int32)	Acquires the standard error stream, which is set to a specified buffer size.

OpenStandardError()

Acquires the standard error stream.

C#

```
public static System.IO.Stream OpenStandardError ();
```

Returns

Stream

The standard error stream.

Examples

The following example is a simple text file viewer that displays the contents of one or more text files to the console. If there are no command line arguments, or if any files passed as command line arguments do not exist, the example calls the SetError method to redirect error information to a file, calls the OpenStandardError method in

the process of reacquiring the standard error stream, and indicates that error information was written to a file.

C#

```
using System;
using System.IO;

public class ViewTextFile
{
   public static void Main()
   {
      String[] args = Environment.GetCommandLineArgs();
      String errorOutput = "";
      // Make sure that there is at least one command line argument.
      if (args.Length <= 1)
         errorOutput += "You must include a filename on the command
line.\n";

      for (int ctr = 1; ctr <= args.GetUpperBound(0); ctr++)  {
         // Check whether the file exists.
         if (! File.Exists(args[ctr])) {
            errorOutput += String.Format("'{0}' does not exist.\n",
args[ctr]);
         }
         else {
            // Display the contents of the file.
            StreamReader sr = new StreamReader(args[ctr]);
            String contents = sr.ReadToEnd();
            sr.Close();
            Console.WriteLine("*****Contents of file '{0}':\n\n",
                              args[ctr]);
            Console.WriteLine(contents);
            Console.WriteLine("*****\n");
         }
      }

      // Check for error conditions.
      if (! String.IsNullOrEmpty(errorOutput)) {
         // Write error information to a file.
         Console.SetError(new StreamWriter(@".\ViewTextFile.Err.txt"));
         Console.Error.WriteLine(errorOutput);
         Console.Error.Close();
         // Reacquire the standard error stream.
         var standardError = new
StreamWriter(Console.OpenStandardError());
         standardError.AutoFlush = true;
         Console.SetError(standardError);
         Console.Error.WriteLine("\nError information written to
ViewTextFile.Err.txt");
      }
   }
}
// If the example is compiled and run with the following command line:
```

```
//     ViewTextFile file1.txt file2.txt
// and neither file1.txt nor file2.txt exist, it displays the
// following output:
//     Error information written to ViewTextFile.Err.txt
// and writes the following text to ViewTextFile.Err.txt:
//     'file1.txt' does not exist.
//     'file2.txt' does not exist.
```

Note that the StreamWriter.AutoFlush property is set to `true` before reacquiring the error stream. This ensures that output will be sent to the console immediately rather than buffered.

Remarks

This method can be used to reacquire the standard error stream after it has been changed by the SetError method.

See also

- Error
- IsErrorRedirected
- SetError(TextWriter)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

OpenStandardError(Int32)

Acquires the standard error stream, which is set to a specified buffer size.

```
C#
```

```
public static System.IO.Stream OpenStandardError (int bufferSize);
```

Parameters

`bufferSize` Int32

This parameter has no effect, but its value must be greater than or equal to zero.

Returns

Stream

The standard error stream.

Exceptions

ArgumentOutOfRangeException

`bufferSize` is less than or equal to zero.

Remarks

This method can be used to reacquire the standard error stream after it has been changed by the SetError method.

See also

- Error
- IsErrorRedirected
- SetError(TextWriter)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.OpenStandardInput Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Acquires the standard input stream.

Overloads

Expand table

OpenStandardInput()	Acquires the standard input stream.
OpenStandardInput(Int32)	Acquires the standard input stream, which is set to a specified buffer size.

OpenStandardInput()

Acquires the standard input stream.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static System.IO.Stream OpenStandardInput ();
```

Returns

Stream

The standard input stream.

Attributes UnsupportedOSPlatformAttribute

Examples

The following example illustrates the use of the `OpenStandardInput` method.

C#

```
using System;
using System.Text;
using System.IO;

public class Decoder {
    public static void Main() {
        Stream inputStream = Console.OpenStandardInput();
        byte[] bytes = new byte[100];
        Console.WriteLine("To decode, type or paste the UTF7 encoded
string and press enter:");
        Console.WriteLine("(Example: \"M+APw-nchen ist wundervoll\")");
        int outputLength = inputStream.Read(bytes, 0, 100);
        char[] chars = Encoding.UTF7.GetChars(bytes, 0, outputLength);
        Console.WriteLine("Decoded string:");
        Console.WriteLine(new string(chars));
    }
}
```

Remarks

This method can be used to reacquire the standard input stream after it has been changed by the SetIn method.

See also

- In
- IsInputRedirected
- SetIn(TextReader)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

OpenStandardInput(Int32)

Acquires the standard input stream, which is set to a specified buffer size.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
public static System.IO.Stream OpenStandardInput (int bufferSize);
```

Parameters

`bufferSize` Int32

This parameter has no effect, but its value must be greater than or equal to zero.

Returns

Stream

The standard input stream.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`bufferSize` is less than or equal to zero.

Remarks

This method can be used to reacquire the standard output stream after it has been changed by the SetIn method.

See also

- In
- IsInputRedirected
- SetIn(TextReader)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.OpenStandardOutput Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Acquires the standard output stream.

Overloads

Expand table

OpenStandardOutput()	Acquires the standard output stream.
OpenStandardOutput(Int32)	Acquires the standard output stream, which is set to a specified buffer size.

OpenStandardOutput()

Acquires the standard output stream.

C#

```
public static System.IO.Stream OpenStandardOutput ();
```

Returns

Stream

The standard output stream.

Examples

The following example illustrates the use of the OpenStandardOutput method. It replaces four consecutive space characters in a string with a tab character. To run it, you must supply two command-line arguments. The first is the name of an existing text file to redirect the standard input stream to. The second is the name of a file to

redirect the standard output stream to. This file need not exist. If it does, its contents will be overwritten.

C#

```
using System;
using System.IO;

public class InsertTabs
{
    private const int tabSize = 4;
    private const string usageText = "Usage: INSERTTABS inputfile.txt
outputfile.txt";
    public static int Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine(usageText);
            return 1;
        }

        try
        {
            // Attempt to open output file.
            using (var writer = new StreamWriter(args[1]))
            {
                using (var reader = new StreamReader(args[0]))
                {
                    // Redirect standard output from the console to the
output file.
                    Console.SetOut(writer);
                    // Redirect standard input from the console to the
input file.
                    Console.SetIn(reader);
                    string line;
                    while ((line = Console.ReadLine()) != null)
                    {
                        string newLine =
line.Replace(("").PadRight(tabSize, ' '), "\t");
                        Console.WriteLine(newLine);
                    }
                }
            }
        }
        catch(IOException e)
        {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return 1;
        }

        // Recover the standard output stream so that a
        // completion message can be displayed.
```

```
        var standardOutput = new
StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
        Console.WriteLine($"INSERTTABS has completed the processing of
{args[0]}.");
        return 0;
    }
}
```

Remarks

This method can be used to reacquire the standard output stream after it has been changed by the SetOut method.

See also

- Out
- IsOutputRedirected
- SetOut(TextWriter)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

OpenStandardOutput(Int32)

Acquires the standard output stream, which is set to a specified buffer size.

C#

```
public static System.IO.Stream OpenStandardOutput (int bufferSize);
```

Parameters

`bufferSize` Int32

This parameter has no effect, but its value must be greater than or equal to zero.

Returns

Stream

The standard output stream.

Exceptions

ArgumentOutOfRangeException

`bufferSize` is less than or equal to zero.

Examples

The following example illustrates the use of the OpenStandardOutput method. It replaces four consecutive space characters in a string with a tab character. To run it, you must supply two command line arguments. The first is the name of an existing text file to redirect the standard input stream to. The second is the name of a file to redirect the standard output stream to. This file need not exist. If it does, its contents will be overwritten.

C#

```
using System;
using System.IO;

public class InsertTabs
{
    private const int tabSize = 4;
    private const string usageText = "Usage: INSERTTABS inputfile.txt
outputfile.txt";
    public static int Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine(usageText);
            return 1;
        }

        try
        {
            // Attempt to open output file.
            using (var writer = new StreamWriter(args[1]))
            {
```

```
                using (var reader = new StreamReader(args[0]))
                {
                    // Redirect standard output from the console to the
output file.
                    Console.SetOut(writer);
                    // Redirect standard input from the console to the
input file.
                    Console.SetIn(reader);
                    string line;
                    while ((line = Console.ReadLine()) != null)
                    {
                        string newLine =
line.Replace("").PadRight(tabSize, ' '), "\t");
                        Console.WriteLine(newLine);
                    }
                }
            }
        }
        catch(IOException e)
        {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return 1;
        }

        // Recover the standard output stream so that a
        // completion message can be displayed.
        var standardOutput = new
StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
        Console.WriteLine($"INSERTTABS has completed the processing of
{args[0]}.");
        return 0;
    }
}
```

Remarks

This method can be used to reacquire the standard output stream after it has been changed by the SetOut method.

See also

- Out
- IsOutputRedirected
- SetOut(TextWriter)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.Read Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Reads the next character from the standard input stream.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
public static int Read ();
```

Returns

Int32

The next character from the input stream, or negative one (-1) if there are currently no more characters to be read.

Attributes UnsupportedOSPlatformAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example demonstrates the Read method.

C#

```
// This example demonstrates the Console.Read() method.
using System;

class Sample
{
    public static void Main()
```

```
    {
    string m1 = "\nType a string of text then press Enter. " +
                "Type '+' anywhere in the text to quit:\n";
    string m2 = "Character '{0}' is hexadecimal 0x{1:x4}.";
    string m3 = "Character     is hexadecimal 0x{0:x4}.";
    char ch;
    int x;
//
    Console.WriteLine(m1);
    do
        {
        x = Console.Read();
        try
            {
            ch = Convert.ToChar(x);
            if (Char.IsWhiteSpace(ch))
               {
               Console.WriteLine(m3, x);
               if (ch == 0x0a)
                   Console.WriteLine(m1);
               }
            else
                {
                    Console.WriteLine(m2, ch, x);
                }
            }
        catch (OverflowException e)
            {
            Console.WriteLine("{0} Value read = {1}.", e.Message, x);
            ch = Char.MinValue;
            Console.WriteLine(m1);
            }
        } while (ch != '+');
    }
}
/*
This example produces the following results:

Type a string of text then press Enter. Type '+' anywhere in the text to
quit:

The quick brown fox.
Character 'T' is hexadecimal 0x0054.
Character 'h' is hexadecimal 0x0068.
Character 'e' is hexadecimal 0x0065.
Character     is hexadecimal 0x0020.
Character 'q' is hexadecimal 0x0071.
Character 'u' is hexadecimal 0x0075.
Character 'i' is hexadecimal 0x0069.
Character 'c' is hexadecimal 0x0063.
Character 'k' is hexadecimal 0x006b.
Character     is hexadecimal 0x0020.
Character 'b' is hexadecimal 0x0062.
Character 'r' is hexadecimal 0x0072.
Character 'o' is hexadecimal 0x006f.
```

```
Character 'w' is hexadecimal 0x0077.
Character 'n' is hexadecimal 0x006e.
Character     is hexadecimal 0x0020.
Character 'f' is hexadecimal 0x0066.
Character 'o' is hexadecimal 0x006f.
Character 'x' is hexadecimal 0x0078.
Character '.' is hexadecimal 0x002e.
Character     is hexadecimal 0x000d.
Character     is hexadecimal 0x000a.

Type a string of text then press Enter. Type '+' anywhere in the text to
quit:

^Z
Value was either too large or too small for a character. Value read = -1.

Type a string of text then press Enter. Type '+' anywhere in the text to
quit:

+
Character '+' is hexadecimal 0x002b.

*/
```

Remarks

The Read method blocks its return while you type input characters; it terminates when you press the Enter key. Pressing Enter appends a platform-dependent line termination sequence to your input (for example, Windows appends a carriage return-linefeed sequence). Subsequent calls to the Read method retrieve your input one character at a time. After the final character is retrieved, Read blocks its return again and the cycle repeats.

> ⓘ **Important**
>
> The **ReadLine** method, or the **KeyAvailable** property and **ReadKey** method are preferable to using the **Read** method.

Note that the method does not return -1 unless you perform one of the following actions:

- Simultaneously press the Control modifier key and Z console key (Ctrl+Z), which signals the end-of-file condition. If you're on Windows, you must also press the Enter console key.

- Press an equivalent key that signals the end-of-file condition, such as the F6 function key in Windows.
- Redirect the input stream to a source, such as a text file, that has an actual end-of-file character.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ReadLine()
- Write(String, Object)
- WriteLine()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.ReadKey Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Obtains the next character or function key pressed by the user.

Overloads

Expand table

ReadKey()	Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.
ReadKey(Boolean)	Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.

ReadKey()

Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static ConsoleKeyInfo ReadKey ();
```

Returns

ConsoleKeyInfo

An object that describes the ConsoleKey constant and Unicode character, if any, that correspond to the pressed console key. The ConsoleKeyInfo object also describes, in

a bitwise combination of ConsoleModifiers values, whether one or more Shift, Alt, or Ctrl modifier keys was pressed simultaneously with the console key.

Attributes UnsupportedOSPlatformAttribute

Exceptions

InvalidOperationException

The In property is redirected from some stream other than the console.

Examples

One of the most common uses of the ReadKey() method is to halt program execution until the user presses a key and the app either terminates or displays an additional window of information. The following example uses the ReadKey() method to wait for the user to press the Enter key before terminating the app.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime dat = DateTime.Now;
      Console.WriteLine("The time: {0:d} at {0:t}", dat);
      TimeZoneInfo tz = TimeZoneInfo.Local;
      Console.WriteLine("The time zone: {0}\n",
                        tz.IsDaylightSavingTime(dat) ?
                           tz.DaylightName : tz.StandardName);
      Console.Write("Press <Enter> to exit... ");
      while (Console.ReadKey().Key != ConsoleKey.Enter) {}
   }
}
// The example displays output like the following:
//     The time: 11/11/2015 at 4:02 PM:
//     The time zone: Pacific Standard Time
```

Note that this overload of the ReadKey method by default echoes any displayable keys that the user presses to the console. To suppress them, call the ReadKey method with an `intercept` argument of `true`.

The following example uses the ReadKey() method to display information about which key the user pressed.

C#

```
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT,
and a console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
         if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
         if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
         if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
         Console.WriteLine(cki.Key.ToString());
       } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Remarks

The ReadKey method waits, that is, blocks on the thread issuing the ReadKey method, until a character or function key is pressed. A character or function key can be pressed in combination with one or more Alt, Ctrl, or Shift modifier keys. However, pressing a modifier key by itself will not cause the ReadKey method to return.

Depending on your application, you might want to use the ReadKey method in conjunction with the KeyAvailable property.

The ReadKey method reads from the keyboard even if the standard input is redirected to a file with the SetIn method.

See also

- ConsoleKey
- KeyAvailable
- TreatControlCAsInput
- CancelKeyPress

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

ReadKey(Boolean)

Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static ConsoleKeyInfo ReadKey (bool intercept);
```

Parameters

`intercept` Boolean

Determines whether to display the pressed key in the console window. `true` to not display the pressed key; otherwise, `false`.

Returns

ConsoleKeyInfo

An object that describes the ConsoleKey constant and Unicode character, if any, that correspond to the pressed console key. The ConsoleKeyInfo object also describes, in a bitwise combination of ConsoleModifiers values, whether one or more Shift, Alt, or Ctrl modifier keys was pressed simultaneously with the console key.

Attributes UnsupportedOSPlatformAttribute

Exceptions

InvalidOperationException

The In property is redirected from some stream other than the console.

Examples

One of the most common uses of the ReadKey method is to halt program execution until the user presses a key and the app either terminates or displays an additional window of information. The following example uses the ReadKey(Boolean) method to wait for the user to press the Enter key before terminating the app. Note that, if the user presses any other key, it is not echoed to the console.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime dat = DateTime.Now;
      Console.WriteLine("The time: {0:d} at {0:t}", dat);
      TimeZoneInfo tz = TimeZoneInfo.Local;
      Console.WriteLine("The time zone: {0}\n",
                        tz.IsDaylightSavingTime(dat) ?
                           tz.DaylightName : tz.StandardName);
      Console.Write("Press <Enter> to exit... ");
      while (Console.ReadKey(true).Key != ConsoleKey.Enter) {}
   }
}
// The example displays output like the following:
```

```
//      The time: 11/11/2015 at 4:02 PM:
//      The time zone: Pacific Standard Time
```

The following example uses the ReadKey(Boolean) method to display information about the key pressed by a user without echoing that key to the console.

```C#
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT,
and a console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do {
         cki = Console.ReadKey(true);
         Console.Write("You pressed ");
         if ((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
         if ((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
         if ((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
         Console.WriteLine("{0} (character '{1}')", cki.Key,
cki.KeyChar);
      } while (cki.Key != ConsoleKey.Escape);
   }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       You pressed CTL+A (character '☺')
//       You pressed C (character 'c')
//       You pressed CTL+C (character '♥')
//       You pressed K (character 'k')
//       You pressed ALT+I (character 'i')
//       You pressed ALT+U (character 'u')
//       You pressed ALT+SHIFT+H (character 'H')
//       You pressed Escape (character '←')
```

Remarks

The ReadKey method waits, that is, blocks on the thread issuing the ReadKey method, until a character or function key is pressed. A character or function key can be pressed in combination with one or more Alt, Ctrl, or Shift modifier keys. However, pressing a modifier key by itself will not cause the ReadKey method to return.

If the `intercept` parameter is `true`, the pressed key is intercepted and not displayed in the console window; otherwise, the pressed key is displayed.

Depending on your application, you might want to use the ReadKey method in conjunction with the KeyAvailable property.

The ReadKey method reads from the keyboard even if the standard input is redirected to a file with the SetIn method.

See also

- ConsoleKey
- KeyAvailable
- TreatControlCAsInput
- CancelKeyPress

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Console.ReadLine Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Reads the next line of characters from the standard input stream.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
public static string? ReadLine ();
```

Returns

String

The next line of characters from the input stream, or `null` if no more lines are available.

Attributes UnsupportedOSPlatformAttribute

Exceptions

IOException

An I/O error occurred.

OutOfMemoryException

There is insufficient memory to allocate a buffer for the returned string.

ArgumentOutOfRangeException

The number of characters in the next line of characters is greater than Int32.MaxValue.

Examples

The following example requires two command line arguments: the name of an existing text file, and the name of a file to write the output to. It opens the existing text file and redirects the standard input from the keyboard to that file. It also redirects the standard

output from the console to the output file. It then uses the Console.ReadLine method to read each line in the file, replaces every sequence of four spaces with a tab character, and uses the Console.WriteLine method to write the result to the output file.

C#

```
using System;
using System.IO;

public class InsertTabs
{
    private const int tabSize = 4;
    private const string usageText = "Usage: INSERTTABS inputfile.txt
outputfile.txt";
    public static int Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine(usageText);
            return 1;
        }

        try
        {
            // Attempt to open output file.
            using (var writer = new StreamWriter(args[1]))
            {
                using (var reader = new StreamReader(args[0]))
                {
                    // Redirect standard output from the console to the
output file.
                    Console.SetOut(writer);
                    // Redirect standard input from the console to the input
file.
                    Console.SetIn(reader);
                    string line;
                    while ((line = Console.ReadLine()) != null)
                    {
                        string newLine = line.Replace(("").PadRight(tabSize,
' '), "\t");
                        Console.WriteLine(newLine);
                    }
                }
            }
        }
        catch(IOException e)
        {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return 1;
        }
```

```
        // Recover the standard output stream so that a
        // completion message can be displayed.
        var standardOutput = new StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
        Console.WriteLine($"INSERTTABS has completed the processing of
{args[0]}.");
        return 0;
    }
}
```

Remarks

The ReadLine method reads a line from the standard input stream. (For the definition of a line, see the paragraph after the following list.) This means that:

- If the standard input device is the keyboard, the ReadLine method blocks until the user presses the **Enter** key.

  One of the most common uses of the ReadLine method is to pause program execution before clearing the console and displaying new information to it, or to prompt the user to press the Enter key before terminating the application. The following example illustrates this.

  C#

  ```
  using System;

  public class Example
  {
     public static void Main()
     {
        Console.Clear();

        DateTime dat = DateTime.Now;

        Console.WriteLine("\nToday is {0:d} at {0:T}.", dat);
        Console.Write("\nPress any key to continue... ");
        Console.ReadLine();
     }
  }
  // The example displays output like the following:
  //     Today is 10/26/2015 at 12:22:22 PM.
  //
  //     Press any key to continue...
  ```

- If standard input is redirected to a file, the ReadLine method reads a line of text from a file. For example, the following is a text file named ReadLine1.txt:

```
This is the first line.
This is the second line.
This is the third line.
This is the fourth line.
```

The following example uses the ReadLine method to read input that is redirected from a file. The read operation terminates when the method returns `null`, which indicates that no lines remain to be read.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      if (! Console.IsInputRedirected) {
         Console.WriteLine("This example requires that input be
redirected from a file.");
         return;
      }

      Console.WriteLine("About to call Console.ReadLine in a loop.");
      Console.WriteLine("----");
      String s;
      int ctr = 0;
      do {
         ctr++;
         s = Console.ReadLine();
         Console.WriteLine("Line {0}: {1}", ctr, s);
      } while (s != null);
      Console.WriteLine("---");
   }
}
// The example displays the following output:
//       About to call Console.ReadLine in a loop.
//       ----
//       Line 1: This is the first line.
//       Line 2: This is the second line.
//       Line 3: This is the third line.
//       Line 4: This is the fourth line.
//       Line 5:
//       ---
```

After compiling the example to an executable named ReadLine1.exe, you can run it from the command line to read the contents of the file and display them to the console. The syntax is:

```
ReadLine1 < ReadLine1.txt
```

A line is defined as a sequence of characters followed by a carriage return (hexadecimal 0x000d), a line feed (hexadecimal 0x000a), or the value of the Environment.NewLine property. The returned string does not contain the terminating character(s). By default, the method reads input from a 256-character input buffer. Because this includes the Environment.NewLine character(s), the method can read lines that contain up to 254 characters. To read longer lines, call the OpenStandardInput(Int32) method.

The ReadLine method executes synchronously. That is, it blocks until a line is read or the `Ctrl+Z` keyboard combination (followed by `Enter` on Windows), is pressed. The In property returns a TextReader object that represents the standard input stream and that has both a synchronous TextReader.ReadLine method and an asynchronous TextReader.ReadLineAsync method. However, when used as the console's standard input stream, the TextReader.ReadLineAsync executes synchronously rather than asynchronously and returns a `Task<String>` only after the read operation has completed.

If this method throws an OutOfMemoryException exception, the reader's position in the underlying Stream object is advanced by the number of characters the method was able to read, but the characters already read into the internal ReadLine buffer are discarded. Since the position of the reader in the stream cannot be changed, the characters already read are unrecoverable, and can be accessed only by reinitializing the TextReader. If the initial position within the stream is unknown or the stream does not support seeking, the underlying Stream also needs to be reinitialized. To avoid such a situation and to produce robust code, you should use the KeyAvailable property and ReadKey method and store the read characters in a pre-allocated buffer.

If the `Ctrl+Z` key combination (followed by `Enter` on Windows) is pressed when the method is reading input from the console, the method returns `null`. This enables the user to prevent further keyboard input when the ReadLine method is called in a loop. The following example illustrates this scenario.

```C#
using System;
```

```
public class Example
{
   public static void Main()
   {
      string line;
      Console.WriteLine("Enter one or more lines of text (press CTRL+Z to
exit):");
      Console.WriteLine();
      do {
         Console.Write("   ");
         line = Console.ReadLine();
         if (line != null)
            Console.WriteLine("      " + line);
      } while (line != null);
   }
}
// The following displays possible output from this example:
//       Enter one or more lines of text (press CTRL+Z to exit):
//
//          This is line #1.
//             This is line #1.
//          This is line #2
//             This is line #2
//          ^Z
//
//       >
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Read()
- Write(String, Object)
- WriteLine()

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.ResetColor Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the foreground and background console colors to their defaults.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void ResetColor ();
```

Attributes UnsupportedOSPlatformAttribute

Exceptions

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

The following example saves the values of the ConsoleColor enumeration to an array and stores the current values of the BackgroundColor and ForegroundColor properties to variables. It then changes the foreground color to each color in the ConsoleColor enumeration except to the color that matches the current background, and it changes the background color to each color in the ConsoleColor enumeration except to the color that matches the current foreground. (If the foreground color is the same as the background color, the text isn't visible.) Finally, it calls the ResetColor method to restore the original console colors.

C#

```
using System;

class Example
{
   public static void Main()
   {
      // Get an array with the values of ConsoleColor enumeration members.
      ConsoleColor[] colors = (ConsoleColor[])
ConsoleColor.GetValues(typeof(ConsoleColor));
      // Save the current background and foreground colors.
      ConsoleColor currentBackground = Console.BackgroundColor;
      ConsoleColor currentForeground = Console.ForegroundColor;

      // Display all foreground colors except the one that matches the
background.
      Console.WriteLine("All the foreground colors except {0}, the
background color:",
                        currentBackground);
      foreach (var color in colors) {
         if (color == currentBackground) continue;

         Console.ForegroundColor = color;
         Console.WriteLine("   The foreground color is {0}.", color);
      }
      Console.WriteLine();
      // Restore the foreground color.
      Console.ForegroundColor = currentForeground;

      // Display each background color except the one that matches the
current foreground color.
      Console.WriteLine("All the background colors except {0}, the
foreground color:",
                        currentForeground);
      foreach (var color in colors) {
         if (color == currentForeground) continue;

         Console.BackgroundColor = color;
         Console.WriteLine("   The background color is {0}.", color);
      }

      // Restore the original console colors.
      Console.ResetColor();
      Console.WriteLine("\nOriginal colors restored...");
   }
}
//The example displays output like the following:
//    All the foreground colors except DarkCyan, the background color:
//       The foreground color is Black.
//       The foreground color is DarkBlue.
//       The foreground color is DarkGreen.
//       The foreground color is DarkRed.
//       The foreground color is DarkMagenta.
//       The foreground color is DarkYellow.
//       The foreground color is Gray.
```

```
//       The foreground color is DarkGray.
//       The foreground color is Blue.
//       The foreground color is Green.
//       The foreground color is Cyan.
//       The foreground color is Red.
//       The foreground color is Magenta.
//       The foreground color is Yellow.
//       The foreground color is White.
//
//    All the background colors except White, the foreground color:
//       The background color is Black.
//       The background color is DarkBlue.
//       The background color is DarkGreen.
//       The background color is DarkCyan.
//       The background color is DarkRed.
//       The background color is DarkMagenta.
//       The background color is DarkYellow.
//       The background color is Gray.
//       The background color is DarkGray.
//       The background color is Blue.
//       The background color is Green.
//       The background color is Cyan.
//       The background color is Red.
//       The background color is Magenta.
//       The background color is Yellow.
//
//    Original colors restored...
```

Remarks

The foreground and background colors are restored to the colors that existed when the current process began. For more information, see the ForegroundColor and BackgroundColor properties.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.SetBufferSize(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the height and width of the screen buffer area to the specified values.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static void SetBufferSize (int width, int height);
```

Parameters

width Int32

The width of the buffer area measured in columns.

height Int32

The height of the buffer area measured in rows.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`height` or `width` is less than or equal to zero.

-or-

`height` or `width` is greater than or equal to Int16.MaxValue.

-or-

`width` is less than WindowLeft + WindowWidth.

-or-

`height` is less than WindowTop + WindowHeight.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The current operating system is not Windows.

Examples

The following example demonstrates the WindowLeft, WindowTop, WindowWidth, WindowHeight, BufferWidth, BufferHeight, and CursorVisible properties; and the SetWindowPosition, SetBufferSize, and ReadKey methods. The example draws a grid pattern in the screen buffer based on the screen buffer width. Then the example moves the console window in response to which of the UP ARROW, DOWN ARROW, LEFT ARROW, or RIGHT ARROW console keys is pressed. The grid pattern helps you see the movement of the console window relative to the screen buffer.

C#

```
// This example demonstrates the Console.WindowLeft and
//                               Console.WindowTop properties.
using System;
using System.Text;
using System.IO;
//
class Sample
{
    public static int saveBufferWidth;
    public static int saveBufferHeight;
    public static int saveWindowHeight;
    public static int saveWindowWidth;
    public static bool saveCursorVisible;
//
    public static void Main()
    {
    string m1 = "1) Press the cursor keys to move the console window.\n" +
                "2) Press any key to begin. When you're finished...\n" +
                "3) Press the Escape key to quit.";
    string g1 = "+----";
    string g2 = "|    ";
    string grid1;
    string grid2;
    StringBuilder sbG1 = new StringBuilder();
```

```
    StringBuilder sbG2 = new StringBuilder();
    ConsoleKeyInfo cki;
    int y;
//
    try
    {
    saveBufferWidth  = Console.BufferWidth;
    saveBufferHeight = Console.BufferHeight;
    saveWindowHeight = Console.WindowHeight;
    saveWindowWidth  = Console.WindowWidth;
    saveCursorVisible = Console.CursorVisible;
//
    Console.Clear();
    Console.WriteLine(m1);
    Console.ReadKey(true);

// Set the smallest possible window size before setting the buffer size.
    Console.SetWindowSize(1, 1);
    Console.SetBufferSize(80, 80);
    Console.SetWindowSize(40, 20);

// Create grid lines to fit the buffer. (The buffer width is 80, but
// this same technique could be used with an arbitrary buffer width.)
    for (y = 0; y < Console.BufferWidth/g1.Length; y++)
        {
        sbG1.Append(g1);
        sbG2.Append(g2);
        }
    sbG1.Append(g1, 0, Console.BufferWidth%g1.Length);
    sbG2.Append(g2, 0, Console.BufferWidth%g2.Length);
    grid1 = sbG1.ToString();
    grid2 = sbG2.ToString();

    Console.CursorVisible = false;
    Console.Clear();
    for (y = 0; y < Console.BufferHeight-1; y++)
        {
        if (y%3 == 0)
            Console.Write(grid1);
        else
            Console.Write(grid2);
        }

    Console.SetWindowPosition(0, 0);
    do
        {
        cki = Console.ReadKey(true);
        switch (cki.Key)
            {
            case ConsoleKey.LeftArrow:
                if (Console.WindowLeft > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft-1, Console.WindowTop);
                break;
            case ConsoleKey.UpArrow:
```

```
                if (Console.WindowTop > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop-1);
                break;
            case ConsoleKey.RightArrow:
                if (Console.WindowLeft < (Console.BufferWidth-
Console.WindowWidth))
                    Console.SetWindowPosition(
                            Console.WindowLeft+1, Console.WindowTop);
                break;
            case ConsoleKey.DownArrow:
                if (Console.WindowTop < (Console.BufferHeight-
Console.WindowHeight))
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop+1);
                break;
            }
        }
    while (cki.Key != ConsoleKey.Escape);  // end do-while
    } // end try
    catch (IOException e)
        {
        Console.WriteLine(e.Message);
        }
    finally
        {
        Console.Clear();
        Console.SetWindowSize(1, 1);
        Console.SetBufferSize(saveBufferWidth, saveBufferHeight);
        Console.SetWindowSize(saveWindowWidth, saveWindowHeight);
        Console.CursorVisible = saveCursorVisible;
        }
    } // end Main
} // end Sample
/*
This example produces results similar to the following:

1) Press the cursor keys to move the console window.
2) Press any key to begin. When you're finished...
3) Press the Escape key to quit.

...

+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.SetCursorPosition(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the position of the cursor.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void SetCursorPosition (int left, int top);
```

Parameters

left Int32

The column position of the cursor. Columns are numbered from left to right starting at 0.

top Int32

The row position of the cursor. Rows are numbered from top to bottom starting at 0.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`left` or `top` is less than zero.

-or-

`left` is greater than or equal to BufferWidth.

-or-

`top` is greater than or equal to BufferHeight.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

Examples

This example demonstrates the CursorLeft and CursorTop properties, and the SetCursorPosition and Clear methods. The example positions the cursor, which determines where the next write will occur, to draw a 5 character by 5 character rectangle using a combination of "+", "|", and "-" strings. Note that the rectangle could be drawn with fewer steps using a combination of other strings.

C#

```
// This example demonstrates the
//     Console.CursorLeft and
//     Console.CursorTop properties, and the
//     Console.SetCursorPosition and
//     Console.Clear methods.
using System;

class Sample
{
    protected static int origRow;
    protected static int origCol;

    protected static void WriteAt(string s, int x, int y)
    {
    try
        {
        Console.SetCursorPosition(origCol+x, origRow+y);
        Console.Write(s);
        }
    catch (ArgumentOutOfRangeException e)
        {
        Console.Clear();
        Console.WriteLine(e.Message);
        }
    }

    public static void Main()
    {
// Clear the screen, then save the top and left coordinates.
    Console.Clear();
    origRow = Console.CursorTop;
```

```
    origCol = Console.CursorLeft;

// Draw the left side of a 5x5 rectangle, from top to bottom.
    WriteAt("+", 0, 0);
    WriteAt("|", 0, 1);
    WriteAt("|", 0, 2);
    WriteAt("|", 0, 3);
    WriteAt("+", 0, 4);

// Draw the bottom side, from left to right.
    WriteAt("-", 1, 4); // shortcut: WriteAt("---", 1, 4)
    WriteAt("-", 2, 4); // ...
    WriteAt("-", 3, 4); // ...
    WriteAt("+", 4, 4);

// Draw the right side, from bottom to top.
    WriteAt("|", 4, 3);
    WriteAt("|", 4, 2);
    WriteAt("|", 4, 1);
    WriteAt("+", 4, 0);

// Draw the top side, from right to left.
    WriteAt("-", 3, 0); // shortcut: WriteAt("---", 1, 0)
    WriteAt("-", 2, 0); // ...
    WriteAt("-", 1, 0); // ...
//
    WriteAt("All done!", 0, 6);
    Console.WriteLine();
    }
}
/*
This example produces the following results:

+---+
|   |
|   |
|   |
+---+

All done!

*/
```

Remarks

Use the SetCursorPosition method to specify where the next write operation in the console window is to begin. If the specified cursor position is outside the area that is currently visible in the console window, the window origin changes automatically to make the cursor visible.

The cursor automatically moves to the next character position each time a character is written to the console window. If the cursor is at the bottom right character position of the console window, the next write operation causes the console window to scroll so the cursor remains visible. If you want to write a character to the bottom right character position without causing the console window to scroll, use the MoveBufferArea method to move a character to that position.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.SetError(TextWriter) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the Error property to the specified TextWriter object.

C#

```
public static void SetError (System.IO.TextWriter newError);
```

Parameters

`newError` TextWriter

A stream that is the new standard error output.

Exceptions

ArgumentNullException

`newError` is `null`.

SecurityException

The caller does not have the required permission.

Examples

The following example shows how to redirect the standard error stream to a file.

C#

```
using System;
using System.IO;
using System.Reflection;

public class RedirectStdErr
{
   public static void Main()
   {
      // Define file to receive error stream.
```

```
        DateTime appStart = DateTime.Now;
        string fn = @"c:\temp\errlog" + appStart.ToString("yyyyMMddHHmm") +
".log";
        TextWriter errStream = new StreamWriter(fn);
        string appName = typeof(RedirectStdErr).Assembly.Location;
        appName = appName.Substring(appName.LastIndexOf('\\') + 1);
        // Redirect standard error stream to file.
        Console.SetError(errStream);
        // Write file header.
        Console.Error.WriteLine("Error Log for Application {0}", appName);
        Console.Error.WriteLine();
        Console.Error.WriteLine("Application started at {0}.", appStart);
        Console.Error.WriteLine();
        //
        // Application code along with error output
        //
        // Close redirected error stream.
        Console.Error.Close();
    }
}
```

Remarks

By default, the Error property is set to the standard error output stream.

A StreamWriter that encapsulates a FileStream can be used to send error messages to a file.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- IsErrorRedirected
- Error

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Console.SetIn(TextReader) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the In property to the specified TextReader object.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void SetIn (System.IO.TextReader newIn);
```

Parameters

newIn TextReader

A stream that is the new standard input.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentNullException

`newIn` is `null`.

SecurityException

The caller does not have the required permission.

Examples

The following example illustrates the use of the SetIn method. It replaces four consecutive space characters in a string with a tab character. To run it, you must supply two command line arguments. The first is the name of an existing text file to redirect the standard input stream to. The second is the name of a file to redirect the standard output stream to. This file need not exist. If it does, its contents will be overwritten.

C#

```
using System;
using System.IO;

public class InsertTabs
{
    private const int tabSize = 4;
    private const string usageText = "Usage: INSERTTABS inputfile.txt
outputfile.txt";
    public static int Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine(usageText);
            return 1;
        }

        try
        {
            // Attempt to open output file.
            using (var writer = new StreamWriter(args[1]))
            {
                using (var reader = new StreamReader(args[0]))
                {
                    // Redirect standard output from the console to the
output file.
                    Console.SetOut(writer);
                    // Redirect standard input from the console to the input
file.
                    Console.SetIn(reader);
                    string line;
                    while ((line = Console.ReadLine()) != null)
                    {
                        string newLine = line.Replace(("").PadRight(tabSize,
' '), "\t");
                        Console.WriteLine(newLine);
                    }
                }
            }
        }
        catch(IOException e)
        {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return 1;
        }

        // Recover the standard output stream so that a
        // completion message can be displayed.
        var standardOutput = new StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
```

```
        Console.WriteLine($"INSERTTABS has completed the processing of
{args[0]}.");
        return 0;
    }
}
```

Remarks

By default, the In property is set to the standard input stream.

A StreamReader that encapsulates a FileStream can be used to receive input from a file.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- TextReader
- In

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.SetOut(TextWriter) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the Out property to target the TextWriter object.

```
C#

public static void SetOut (System.IO.TextWriter newOut);
```

Parameters

newOut TextWriter

A text writer to be used as the new standard output.

Exceptions

ArgumentNullException

`newOut` is `null`.

SecurityException

The caller does not have the required permission.

Examples

The following example illustrates the use of the SetOut method. It replaces four consecutive space characters in a string with a tab character. To run it, you must supply two command line arguments. The first is the name of an existing text file to redirect the standard input stream to. The second is the name of a file to redirect the standard output stream to. This file need not exist. If it does, its contents will be overwritten.

```
C#

using System;
using System.IO;

public class InsertTabs
```

```
{
    private const int tabSize = 4;
    private const string usageText = "Usage: INSERTTABS inputfile.txt
outputfile.txt";
    public static int Main(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine(usageText);
            return 1;
        }

        try
        {
            // Attempt to open output file.
            using (var writer = new StreamWriter(args[1]))
            {
                using (var reader = new StreamReader(args[0]))
                {
                    // Redirect standard output from the console to the
output file.
                    Console.SetOut(writer);
                    // Redirect standard input from the console to the input
file.
                    Console.SetIn(reader);
                    string line;
                    while ((line = Console.ReadLine()) != null)
                    {
                        string newLine = line.Replace(("").PadRight(tabSize,
' '), "\t");
                        Console.WriteLine(newLine);
                    }
                }
            }
        }
        catch(IOException e)
        {
            TextWriter errorWriter = Console.Error;
            errorWriter.WriteLine(e.Message);
            errorWriter.WriteLine(usageText);
            return 1;
        }

        // Recover the standard output stream so that a
        // completion message can be displayed.
        var standardOutput = new StreamWriter(Console.OpenStandardOutput());
        standardOutput.AutoFlush = true;
        Console.SetOut(standardOutput);
        Console.WriteLine($"INSERTTABS has completed the processing of
{args[0]}.");
        return 0;
    }
}
```

Remarks

By default, the Out property is set to the standard output stream.

A StreamWriter that encapsulates a FileStream can be used to send output to a file. For example:

```C#
Console.WriteLine("Hello World");
FileStream fs = new FileStream("Test.txt", FileMode.Create);
// First, save the standard output.
TextWriter tmp = Console.Out;
StreamWriter sw = new StreamWriter(fs);
Console.SetOut(sw);
Console.WriteLine("Hello file");
Console.SetOut(tmp);
Console.WriteLine("Hello World");
sw.Close();
```

The actual object returned by Out may be a synchronized wrapper around the provided text writer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- TextWriter
- Out

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Console.SetWindowPosition(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the position of the console window relative to the screen buffer.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static void SetWindowPosition (int left, int top);
```

Parameters

left Int32

The column position of the upper left corner of the console window.

top Int32

The row position of the upper left corner of the console window.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`left` or `top` is less than zero.

-or-

`left` + WindowWidth is greater than BufferWidth.

-or-

`top` + WindowHeight is greater than BufferHeight.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The current operating system is not Windows.

Examples

The following example demonstrates the WindowLeft, WindowTop, WindowWidth, WindowHeight, BufferWidth, BufferHeight, and CursorVisible properties; and the SetWindowPosition, SetBufferSize, and ReadKey methods. The example draws a grid pattern in the screen buffer based on the screen buffer width. Then the example moves the console window in response to which of the UP ARROW, DOWN ARROW, LEFT ARROW, or RIGHT ARROW console keys is pressed. The grid pattern helps you see the movement of the console window relative to the screen buffer.

C#

```
// This example demonstrates the Console.WindowLeft and
//                               Console.WindowTop properties.
using System;
using System.Text;
using System.IO;
//
class Sample
{
    public static int saveBufferWidth;
    public static int saveBufferHeight;
    public static int saveWindowHeight;
    public static int saveWindowWidth;
    public static bool saveCursorVisible;
//
    public static void Main()
    {
    string m1 = "1) Press the cursor keys to move the console window.\n" +
                "2) Press any key to begin. When you're finished...\n" +
                "3) Press the Escape key to quit.";
    string g1 = "+----";
    string g2 = "|    ";
    string grid1;
    string grid2;
    StringBuilder sbG1 = new StringBuilder();
    StringBuilder sbG2 = new StringBuilder();
    ConsoleKeyInfo cki;
    int y;
//
```

```
    try
    {
    saveBufferWidth  = Console.BufferWidth;
    saveBufferHeight = Console.BufferHeight;
    saveWindowHeight = Console.WindowHeight;
    saveWindowWidth  = Console.WindowWidth;
    saveCursorVisible = Console.CursorVisible;
//
    Console.Clear();
    Console.WriteLine(m1);
    Console.ReadKey(true);

// Set the smallest possible window size before setting the buffer size.
    Console.SetWindowSize(1, 1);
    Console.SetBufferSize(80, 80);
    Console.SetWindowSize(40, 20);

// Create grid lines to fit the buffer. (The buffer width is 80, but
// this same technique could be used with an arbitrary buffer width.)
    for (y = 0; y < Console.BufferWidth/g1.Length; y++)
        {
        sbG1.Append(g1);
        sbG2.Append(g2);
        }
    sbG1.Append(g1, 0, Console.BufferWidth%g1.Length);
    sbG2.Append(g2, 0, Console.BufferWidth%g2.Length);
    grid1 = sbG1.ToString();
    grid2 = sbG2.ToString();

    Console.CursorVisible = false;
    Console.Clear();
    for (y = 0; y < Console.BufferHeight-1; y++)
        {
        if (y%3 == 0)
            Console.Write(grid1);
        else
            Console.Write(grid2);
        }

    Console.SetWindowPosition(0, 0);
    do
        {
        cki = Console.ReadKey(true);
        switch (cki.Key)
            {
            case ConsoleKey.LeftArrow:
                if (Console.WindowLeft > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft-1, Console.WindowTop);
                break;
            case ConsoleKey.UpArrow:
                if (Console.WindowTop > 0)
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop-1);
                break;
```

```
            case ConsoleKey.RightArrow:
                if (Console.WindowLeft < (Console.BufferWidth-
Console.WindowWidth))
                    Console.SetWindowPosition(
                            Console.WindowLeft+1, Console.WindowTop);
                break;
            case ConsoleKey.DownArrow:
                if (Console.WindowTop < (Console.BufferHeight-
Console.WindowHeight))
                    Console.SetWindowPosition(
                            Console.WindowLeft, Console.WindowTop+1);
                break;
            }
        }
    while (cki.Key != ConsoleKey.Escape);  // end do-while
    } // end try
    catch (IOException e)
        {
        Console.WriteLine(e.Message);
        }
    finally
        {
        Console.Clear();
        Console.SetWindowSize(1, 1);
        Console.SetBufferSize(saveBufferWidth, saveBufferHeight);
        Console.SetWindowSize(saveWindowWidth, saveWindowHeight);
        Console.CursorVisible = saveCursorVisible;
        }
    } // end Main
} // end Sample
/*
This example produces results similar to the following:

1) Press the cursor keys to move the console window.
2) Press any key to begin. When you're finished...
3) Press the Escape key to quit.

...

+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-
|    |    |    |
|    |    |    |
+----+----+----+-

*/
```

Remarks

The operating system window displays the console window, and the console window displays a portion of the screen buffer. The SetWindowPosition method affects the position of the console window relative to the screen buffer, but does not affect the position of the operating system window relative to the desktop.

The console and operating system windows generally do not affect each other. However, if the screen buffer cannot be displayed in the current boundaries of the console window, the operating system automatically appends scroll bars to the operating system window. In that case, moving the operating system window scroll bars affects the position of the console window, and moving the console window with the SetWindowPosition method affects the position of the operating system window scroll bars.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.SetWindowSize(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Sets the height and width of the console window to the specified values.

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static void SetWindowSize (int width, int height);
```

Parameters

width Int32

The width of the console window measured in columns.

height Int32

The height of the console window measured in rows.

Attributes UnsupportedOSPlatformAttribute

Exceptions

ArgumentOutOfRangeException

`width` or `height` is less than or equal to zero.

-or-

`width` plus WindowLeft or `height` plus WindowTop is greater than or equal to Int16.MaxValue.

-or-

`width` or `height` is greater than the largest possible window width or height for the current screen resolution and console font.

SecurityException

The user does not have permission to perform this action.

IOException

An I/O error occurred.

PlatformNotSupportedException

The current operating system is not Windows.

Examples

This example demonstrates the SetWindowSize method, and the WindowWidth and WindowHeight properties. You must run the example to see the full effect of changing the console window size.

The example reports the dimensions of a console window set to 85 columns and 43 rows, then waits for a key to be pressed. When any key is pressed, the dimensions of the console window are halved, the new dimensions are reported, and the example waits for another key press. Finally, when any key is pressed, the console window is restored to its original dimensions and the example terminates.

C#

```
// This example demonstrates the Console.SetWindowSize method,
//                            the Console.WindowWidth property,
//                        and the Console.WindowHeight property.
using System;

class Sample
{
    public static void Main()
    {
    int origWidth, width;
    int origHeight, height;
    string m1 = "The current window width is {0}, and the " +
                "current window height is {1}.";
    string m2 = "The new window width is {0}, and the new " +
                "window height is {1}.";
    string m4 = "  (Press any key to continue...)";
//
// Step 1: Get the current window dimensions.
//
    origWidth  = Console.WindowWidth;
```

```
    origHeight = Console.WindowHeight;
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 2: Cut the window to 1/4 its original size.
//
    width  = origWidth/2;
    height = origHeight/2;
    Console.SetWindowSize(width, height);
    Console.WriteLine(m2, Console.WindowWidth,
                          Console.WindowHeight);
    Console.WriteLine(m4);
    Console.ReadKey(true);
//
// Step 3: Restore the window to its original size.
//
    Console.SetWindowSize(origWidth, origHeight);
    Console.WriteLine(m1, Console.WindowWidth,
                          Console.WindowHeight);
    }
}
/*
This example produces the following results:

The current window width is 85, and the current window height is 43.
  (Press any key to continue...)
The new window width is 42, and the new window height is 21.
  (Press any key to continue...)
The current window width is 85, and the current window height is 43.

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Console.Write Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Writes the text representation of the specified value or values to the standard output stream.

Overloads

Expand table

Write(String, Object, Object, Object)	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write(String, Object, Object)	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write(String, Object[])	Writes the text representation of the specified array of objects to the standard output stream using the specified format information.
Write(String, Object)	Writes the text representation of the specified object to the standard output stream using the specified format information.
Write(UInt64)	Writes the text representation of the specified 64-bit unsigned integer value to the standard output stream.
Write(UInt32)	Writes the text representation of the specified 32-bit unsigned integer value to the standard output stream.
Write(String)	Writes the specified string value to the standard output stream.
Write(Char[], Int32, Int32)	Writes the specified subarray of Unicode characters to the standard output stream.
Write(Object)	Writes the text representation of the specified object to the standard output stream.
Write(Single)	Writes the text representation of the specified single-precision floating-point value to the standard output stream.
Write(Char)	Writes the specified Unicode character value to the standard output

	stream.
Write(Char[])	Writes the specified array of Unicode characters to the standard output stream.
Write(Boolean)	Writes the text representation of the specified Boolean value to the standard output stream.
Write(Double)	Writes the text representation of the specified double-precision floating-point value to the standard output stream.
Write(Int32)	Writes the text representation of the specified 32-bit signed integer value to the standard output stream.
Write(Int64)	Writes the text representation of the specified 64-bit signed integer value to the standard output stream.
Write(Decimal)	Writes the text representation of the specified Decimal value to the standard output stream.

Write(String, Object, Object, Object)

Writes the text representation of the specified objects to the standard output stream using the specified format information.

C#

```
public static void Write (string format, object? arg0, object? arg1,
object? arg2);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to write using `format`.

`arg1` Object

The second object to write using `format`.

`arg2` Object

The third object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example uses the `WriteLine` method to demonstrate the standard formatting specifiers for numbers, dates, and enumerations.

C#

```
// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . . . . . {0:D}\n" +
            "(E) Scientific: . . . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . . . {1:F}\n" +
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . . . . . {0:N}\n" +
            "(P) Percent:. . . . . . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
```

```
            "(d) Short date: . . . . . {0:d}\n" +
            "(D) Long date:. . . . . {0:D}\n" +
            "(t) Short time: . . . . . {0:t}\n" +
            "(T) Long time:. . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . {0:R}\n" +
            "(s) Sortable: . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
            "(Y) Year: . . . . . {0:Y}\n",
            thisDate);

        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . {0:G}\n" +
            "    (default):. . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . ($123.00)
(D) Decimal:. . . . . -123
(E) Scientific: . . . . . -1.234500E+002
(F) Fixed point:. . . . . -123.45
(G) General:. . . . . -123
    (default):. . . . . -123 (default = 'G')
(N) Number: . . . . . -123.00
(P) Percent:. . . . . -12,345.00 %
(R) Round-trip: . . . . . -123.45
(X) Hexadecimal:. . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . 6/26/2004
(D) Long date:. . . . . Saturday, June 26, 2004
(t) Short time: . . . . . 8:11 PM
(T) Long time:. . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . June 26
(R) RFC1123:. . . . . Sat, 26 Jun 2004 20:11:04 GMT
```

```
(s) Sortable: . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . Green
    (default):. . . . . Green (default = 'G')
(F) Flags:. . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . 00000003

*/
```

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{`*index*[`,`*alignment*][`:`*formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Read()
- ReadLine()
- WriteLine()
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(String, Object, Object)

Writes the text representation of the specified objects to the standard output stream using the specified format information.

C#

```
public static void Write (string format, object? arg0, object? arg1);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to write using `format`.

`arg1` Object

The second object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example uses the `WriteLine` method to demonstrate the standard formatting specifiers for numbers, dates, and enumerations.

C#

```
// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . . . . . {0:D}\n" +
            "(E) Scientific: . . . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . . . {1:F}\n" +
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . . . . . {0:N}\n" +
            "(P) Percent:. . . . . . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
            "(d) Short date: . . . . . . . {0:d}\n" +
            "(D) Long date:. . . . . . . . {0:D}\n" +
            "(t) Short time: . . . . . . . {0:t}\n" +
            "(T) Long time:. . . . . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . . . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . . . . . {0:R}\n" +
            "(s) Sortable: . . . . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
```

```
            "(Y) Year: . . . . . . . . . . {0:Y}\n",
            thisDate);

        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . . . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . . . . ($123.00)
(D) Decimal:. . . . . . . . . -123
(E) Scientific: . . . . . . . -1.234500E+002
(F) Fixed point:. . . . . . . -123.45
(G) General:. . . . . . . . . -123
    (default):. . . . . . . . -123 (default = 'G')
(N) Number: . . . . . . . . . -123.00
(P) Percent:. . . . . . . . . -12,345.00 %
(R) Round-trip: . . . . . . . -123.45
(X) Hexadecimal:. . . . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . . . 6/26/2004
(D) Long date:. . . . . . . . Saturday, June 26, 2004
(t) Short time: . . . . . . . 8:11 PM
(T) Long time:. . . . . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . . . . . . June 26
(R) RFC1123:. . . . . . . . . Sat, 26 Jun 2004 20:11:04 GMT
(s) Sortable: . . . . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . . . . . Green
    (default):. . . . . . . . Green (default = 'G')
(F) Flags:. . . . . . . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . . . 00000003

*/
```

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{`*index*[`,`*alignment*][`:`*formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Read()
- ReadLine()
- WriteLine()
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(String, Object[])

Writes the text representation of the specified array of objects to the standard output stream using the specified format information.

```
C#

public static void Write (string format, params object?[]? arg);
```

Parameters

format String

A composite format string.

arg Object[]

An array of objects to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` or `arg` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example defines a `Person` class with a number of properties that provide information about a person. Its `GetDescription` method returns an array that contains all of the property values except one. The example then uses the array returned by the `GetDescription` method to display the values of the Person object.

```
C#

using System;

public class Person
{
   public String Name { get; set; }
   public DateTime BirthDate  { get; set; }
   public Double Height { get; set; }
   public Double Weight { get; set; }
```

```
   public Char Gender { get; set; }
   public String Remarks { get; set; }

   public object[] GetDescription()
   {
      return new object[] { Name, Gender, Height, Weight, BirthDate};
   }
}

public class Example
{
   public static void Main()
   {
      var p1 = new Person() { Name = "John", Gender = 'M',
                              BirthDate = new DateTime(1992, 5, 10),
                              Height = 73.5, Weight = 207 };
      p1.Remarks = "Client since 1/3/2012";
      Console.Write("{0}: {1}, born {4:d}  Height {2} inches, Weight {3}
lbs  ",
                    p1.GetDescription());
      if (String.IsNullOrEmpty(p1.Remarks))
         Console.WriteLine();
      else
         Console.WriteLine("{1}Remarks: {0}", p1.Remarks,
                           Console.CursorLeft + p1.Remarks.Length + 10 >
Console.WindowWidth ?
                              "\n   " : "");
   }
}
// The example displays the following output:
//    John: M, born 5/10/1992  Height 73.5 inches, Weight 207 lbs
Remarks: Client since 1/3/2012
```

Note that the example calls the Write(String, Object[]) method rather than the WriteLine(String, Object[]) method because it attempts to display the value of the `Person.Remarks` property on the same line. To do this, it examines the value of the CursorLeft and WindowWidth properties to determine whether there is enough space for the remark to fit. If there is, it displays the line. If not, it writes a line, indents three spaces, and displays the remark.

The following example is identical to the first, except that it supplies a five-item list as the `arg` argument instead of a parameter array.

C#

```
using System;

public class Person
{
   public String Name { get; set; }
   public DateTime BirthDate  { get; set; }
```

```
    public Double Height { get; set; }
    public Double Weight { get; set; }
    public Char Gender { get; set; }
    public String Remarks { get; set; }

    public object[] GetDescription()
    {
       return new object[] { Name, Gender, Height, Weight, BirthDate};
    }
 }

 public class Example
 {
    public static void Main()
    {
       var p1 = new Person() { Name = "John", Gender = 'M',
                               BirthDate = new DateTime(1992, 5, 10),
                               Height = 73.5, Weight = 207 };
       p1.Remarks = "Client since 1/3/2012";
       Console.Write("{0}: {1}, born {2:d}  Height {3} inches, Weight {4}
 lbs  ",
                      p1.Name, p1.Gender, p1.BirthDate, p1.Height,
 p1.Weight);
       if (String.IsNullOrEmpty(p1.Remarks))
          Console.WriteLine();
       else
          Console.WriteLine("{1}Remarks: {0}", p1.Remarks,
                            Console.CursorLeft + p1.Remarks.Length + 10 >
 Console.WindowWidth ?
                               "\n    " : "");
    }
 }
 // The example displays the following output:
 //    John: M, born 5/10/1992  Height 73.5 inches, Weight 207 lbs
 Remarks: Client since 1/3/2012
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{`*index*[`,`*alignment*][`:`*formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an

optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

The `arg` parameter is a parameter array. Arguments can be passed to the method either as an array or as list of five or more items. The examples illustrate both forms of method call.

Notes to Callers

This method is not called by C++ code. The C++ compiler resolves calls to System.Console.Write that include a string and a list of four or more object parameters as a call to Write(String, Object, Object, Object, Object). It resolves calls to System.Console.Write that include a string and an object array as a call to Write(String, Object).

See also

- Read()
- ReadLine()
- WriteLine()
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(String, Object)

Writes the text representation of the specified object to the standard output stream using the specified format information.

C#

```
public static void Write (string format, object? arg0);
```

Parameters

`format` String

A composite format string.

`arg0` Object

An object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example uses the `WriteLine` method to demonstrate the standard formatting specifiers for numbers, dates, and enumerations.

C#

```
// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . {0:D}\n" +
            "(E) Scientific: . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . {1:F}\n" +
            "(G) General:. . . . . {0:G}\n" +
            "    (default):. . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . {0:N}\n" +
            "(P) Percent:. . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . {0:X}\n",
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
            "(d) Short date: . . . . . {0:d}\n" +
            "(D) Long date:. . . . . {0:D}\n" +
            "(t) Short time: . . . . . {0:t}\n" +
            "(T) Long time:. . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . {0:R}\n" +
            "(s) Sortable: . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
            "(Y) Year: . . . . . {0:Y}\n",
            thisDate);
```

```
        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . . . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . . . . ($123.00)
(D) Decimal:. . . . . . . . . -123
(E) Scientific: . . . . . . . -1.234500E+002
(F) Fixed point:. . . . . . . -123.45
(G) General:. . . . . . . . . -123
    (default):. . . . . . . . -123 (default = 'G')
(N) Number: . . . . . . . . . -123.00
(P) Percent:. . . . . . . . . -12,345.00 %
(R) Round-trip: . . . . . . . -123.45
(X) Hexadecimal:. . . . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . . . 6/26/2004
(D) Long date:. . . . . . . . Saturday, June 26, 2004
(t) Short time: . . . . . . . 8:11 PM
(T) Long time:. . . . . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . . . . . . June 26
(R) RFC1123:. . . . . . . . . Sat, 26 Jun 2004 20:11:04 GMT
(s) Sortable: . . . . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . . . . . Green
    (default):. . . . . . . . Green (default = 'G')
(F) Flags:. . . . . . . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . . . 00000003

*/
```

The following example illustrates the use of the Write method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

This method uses the composite formatting feature of the .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{` *index*[`,` *alignment*][`:` *formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Read()
- ReadLine()
- WriteLine()
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(UInt64)

Important

This API is not CLS-compliant.

Writes the text representation of the specified 64-bit unsigned integer value to the standard output stream.

C#

```
[System.CLSCompliant(false)]
public static void Write (ulong value);
```

Parameters

value UInt64

The value to write.

Attributes CLSCompliantAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
```

```
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the UInt64.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(UInt32)

> Important
>
> This API is not CLS-compliant.

Writes the text representation of the specified 32-bit unsigned integer value to the standard output stream.

C#

```
[System.CLSCompliant(false)]
public static void Write (uint value);
```

Parameters

`value` UInt32

The value to write.

Attributes CLSCompliantAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
```

```
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the UInt32.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(String)

Writes the specified string value to the standard output stream.

C#

```
public static void Write (string? value);
```

Parameters

`value` String

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

If value is `null`, nothing is written to the standard output stream.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Char[], Int32, Int32)

Writes the specified subarray of Unicode characters to the standard output stream.

C#

```
public static void Write (char[] buffer, int index, int count);
```

Parameters

`buffer` Char[]

An array of Unicode characters.

`index` Int32

The starting position in `buffer`.

`count` Int32

The number of characters to write.

Exceptions

ArgumentNullException

`buffer` is `null`.

ArgumentOutOfRangeException

`index` or `count` is less than zero.

ArgumentException

`index` plus `count` specify a position that is not within `buffer`.

IOException

An I/O error occurred.

Remarks

This method writes `count` characters starting at position `index` of `buffer` to the standard output stream.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Object)

Writes the text representation of the specified object to the standard output stream.

C#

```
public static void Write (object? value);
```

Parameters

`value` Object

The value to write, or `null`.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

If `value` is `null`, nothing is written and no exception is thrown. Otherwise, the `ToString` method of `value` is called to produce its string representation, and the resulting string is written to the standard output stream.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Single)

Writes the text representation of the specified single-precision floating-point value to the standard output stream.

C#

```
public static void Write (float value);
```

Parameters

`value` Single

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the Single.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Char)

Writes the specified Unicode character value to the standard output stream.

C#

```
public static void Write (char value);
```

Parameters

`value` Char

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
```

```
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Char[])

Writes the specified array of Unicode characters to the standard output stream.

C#

```
public static void Write (char[]? buffer);
```

Parameters

`buffer` Char[]

A Unicode character array.

Exceptions

IOException

An I/O error occurred.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Boolean)

Writes the text representation of the specified Boolean value to the standard output stream.

C#

```
public static void Write (bool value);
```

Parameters

`value` Boolean

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling Boolean.ToString, which outputs either Boolean.TrueString or Boolean.FalseString.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Double)

Writes the text representation of the specified double-precision floating-point value to the standard output stream.

C#

```
public static void Write (double value);
```

Parameters

`value` Double

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the Double.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Int32)

Writes the text representation of the specified 32-bit signed integer value to the standard output stream.

C#

```
public static void Write (int value);
```

Parameters

`value` Int32

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
```

```
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the Int32.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Int64)

Writes the text representation of the specified 64-bit signed integer value to the standard output stream.

C#

```
public static void Write (long value);
```

Parameters

`value` Int64

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
```

```
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling the Int64.ToString method.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Write(Decimal)

Writes the text representation of the specified Decimal value to the standard output stream.

C#

```
public static void Write (decimal value);
```

Parameters

`value` Decimal

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example illustrates the use of the `Write` method.

C#

```
public class FormatConverter
{
    public static void Main(string[] args)
    {
        string lineInput;
        while ((lineInput = Console.ReadLine()) != null)
        {
            string[] fields = lineInput.Split(new char[] {'\t'});
            bool isFirstField = true;
            foreach (var item in fields)
            {
                if (isFirstField)
                    isFirstField = false;
                else
                    Console.Write(',');

                // If the field represents a boolean, replace with a
numeric representation.
                bool itemBool;
                if (Boolean.TryParse(item, out itemBool))
                    Console.Write(Convert.ToByte(itemBool));
                else
                    Console.Write(item);
            }
            Console.WriteLine();
        }
    }
}
```

Remarks

The text representation of `value` is produced by calling Decimal.ToString.

See also

- Read()
- ReadLine()
- WriteLine()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.WriteLine Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Writes the specified data, followed by the current line terminator, to the standard output stream.

Overloads

Expand table

WriteLine(String, Object, Object)	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(String)	Writes the specified string value, followed by the current line terminator, to the standard output stream.
WriteLine(Char[], Int32, Int32)	Writes the specified subarray of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine(String, Object[])	Writes the text representation of the specified array of objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(String, Object)	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(UInt64)	Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the standard output stream.
WriteLine(UInt32)	Writes the text representation of the specified 32-bit unsigned integer value, followed by the current line terminator, to the standard output stream.
WriteLine(Single)	Writes the text representation of the specified single-precision floating-point value, followed by the current line terminator, to the standard output stream.

WriteLine(Decimal)	Writes the text representation of the specified Decimal value, followed by the current line terminator, to the standard output stream.
WriteLine(Int64)	Writes the text representation of the specified 64-bit signed integer value, followed by the current line terminator, to the standard output stream.
WriteLine(Int32)	Writes the text representation of the specified 32-bit signed integer value, followed by the current line terminator, to the standard output stream.
WriteLine(Double)	Writes the text representation of the specified double-precision floating-point value, followed by the current line terminator, to the standard output stream.
WriteLine(Char[])	Writes the specified array of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine(Char)	Writes the specified Unicode character, followed by the current line terminator, value to the standard output stream.
WriteLine(Boolean)	Writes the text representation of the specified Boolean value, followed by the current line terminator, to the standard output stream.
WriteLine()	Writes the current line terminator to the standard output stream.
WriteLine(String, Object, Object, Object)	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine(Object)	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream.

Remarks

The default line terminator is a string whose value is a carriage return followed by a line feed ("\r\n" in C#, or `vbCrLf` in Visual Basic). You can change the line terminator by setting the TextWriter.NewLine property of the Out property to another string.

WriteLine(String, Object, Object)

Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.

```
C#

public static void WriteLine (string format, object? arg0, object? arg1);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to write using `format`.

`arg1` Object

The second object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example demonstrates the standard formatting specifiers for numbers, dates, and enumerations.

```
C#

// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
```

```
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . . . . . {0:D}\n" +
            "(E) Scientific: . . . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . . . {1:F}\n" +
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . . . . . {0:N}\n" +
            "(P) Percent:. . . . . . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
            "(d) Short date: . . . . . . . {0:d}\n" +
            "(D) Long date:. . . . . . . . {0:D}\n" +
            "(t) Short time: . . . . . . . {0:t}\n" +
            "(T) Long time:. . . . . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . . . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . . . . . {0:R}\n" +
            "(s) Sortable: . . . . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
            "(Y) Year: . . . . . . . . . . {0:Y}\n",
            thisDate);

        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . . . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . . . . ($123.00)
```

```
(D) Decimal:. . . . . -123
(E) Scientific: . . . . . -1.234500E+002
(F) Fixed point:. . . . . -123.45
(G) General:. . . . . -123
    (default):. . . . . -123 (default = 'G')
(N) Number: . . . . . -123.00
(P) Percent:. . . . . -12,345.00 %
(R) Round-trip: . . . . . -123.45
(X) Hexadecimal:. . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . 6/26/2004
(D) Long date:. . . . . Saturday, June 26, 2004
(t) Short time: . . . . . 8:11 PM
(T) Long time:. . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . June 26
(R) RFC1123:. . . . . Sat, 26 Jun 2004 20:11:04 GMT
(s) Sortable: . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . Green
    (default):. . . . . Green (default = 'G')
(F) Flags:. . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . 00000003

*/
```

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
```

```
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{` *index*[`,` *alignment*][`:` *formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(String)

Writes the specified string value, followed by the current line terminator, to the standard output stream.

```
C#
```

```
public static void WriteLine (string? value);
```

Parameters

`value` String

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The example changes the line terminator from its default value of "\r\n" or `vbCrLf` to "\r\n\r\n" or `vbCrLf` + `vbCrLf`. It then calls the WriteLine() and WriteLine(String) methods to display output to the console.

C#

```
string[] lines = { "This is the first line.",
                   "This is the second line." };
// Output the lines using the default newline sequence.
Console.WriteLine("With the default new line characters:");
Console.WriteLine();
foreach (string line in lines)
   Console.WriteLine(line);

Console.WriteLine();

// Redefine the newline characters to double space.
Console.Out.NewLine = "\r\n\r\n";
// Output the lines using the new newline sequence.
Console.WriteLine("With redefined new line characters:");
Console.WriteLine();
foreach (string line in lines)
   Console.WriteLine(line);

// The example displays the following output:
//       With the default new line characters:
//
//       This is the first line.
//       This is the second line.
//
//       With redefined new line characters:
//
//
```

```
//
//       This is the first line.
//
//       This is the second line.
```

Remarks

If value is `null`, only the line terminator is written to the standard output stream.

For more information about the line terminator, see the Remarks section of the WriteLine() method.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Char[], Int32, Int32)

Writes the specified subarray of Unicode characters, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (char[] buffer, int index, int count);
```

Parameters

`buffer` Char[]

An array of Unicode characters.

`index` Int32

The starting position in `buffer`.

`count` Int32

The number of characters to write.

Exceptions

ArgumentNullException

`buffer` is `null`.

ArgumentOutOfRangeException

`index` or `count` is less than zero.

ArgumentException

`index` plus `count` specify a position that is not within `buffer`.

IOException

An I/O error occurred.

Remarks

This method writes `count` characters starting at position `index` of `buffer` to the standard output stream.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(String, Object[])

Writes the text representation of the specified array of objects, followed by the current line terminator, to the standard output stream using the specified format information.

C#

```
public static void WriteLine (string format, params object?[]? arg);
```

Parameters

`format` String

A composite format string.

`arg` Object[]

An array of objects to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` or `arg` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example demonstrates the standard formatting specifiers for numbers, dates, and enumerations.

C#

```
// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . . . . . {0:D}\n" +
            "(E) Scientific: . . . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . . . {1:F}\n" +
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . . . . . {0:N}\n" +
            "(P) Percent:. . . . . . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
            "(d) Short date: . . . . . . . {0:d}\n" +
            "(D) Long date:. . . . . . . . {0:D}\n" +
            "(t) Short time: . . . . . . . {0:t}\n" +
            "(T) Long time:. . . . . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . . . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . . . . . {0:R}\n" +
            "(s) Sortable: . . . . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
            "(Y) Year: . . . . . . . . . . {0:Y}\n",
            thisDate);
```

```
        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . . . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . . . . ($123.00)
(D) Decimal:. . . . . . . . . -123
(E) Scientific: . . . . . . . -1.234500E+002
(F) Fixed point:. . . . . . . -123.45
(G) General:. . . . . . . . . -123
    (default):. . . . . . . . -123 (default = 'G')
(N) Number: . . . . . . . . . -123.00
(P) Percent:. . . . . . . . . -12,345.00 %
(R) Round-trip: . . . . . . . -123.45
(X) Hexadecimal:. . . . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . . . 6/26/2004
(D) Long date:. . . . . . . . Saturday, June 26, 2004
(t) Short time: . . . . . . . 8:11 PM
(T) Long time:. . . . . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . . . . . . June 26
(R) RFC1123:. . . . . . . . . Sat, 26 Jun 2004 20:11:04 GMT
(s) Sortable: . . . . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . . . . . Green
    (default):. . . . . . . . Green (default = 'G')
(F) Flags:. . . . . . . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . . . 00000003

*/
```

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the

parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{` *index*[`,` *alignment*][`:` *formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

Notes to Callers

This method is not called by C++ code. The C++ compiler resolves calls to System.Console.WriteLine that include a string and a list of four or more object parameters as a call to WriteLine(String, Object, Object, Object, Object). It resolves calls to System.Console.WriteLine that include a string and an object array as a call to WriteLine(String, Object).

See also

- Read()
- ReadLine()
- Write(String, Object)
- Formatting Types in .NET

- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(String, Object)

Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream using the specified format information.

C#

```
public static void WriteLine (string format, object? arg0);
```

Parameters

`format` String

A composite format string.

`arg0` Object

An object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example calls the WriteLine(String, Object) method to display five randomly generated Boolean values.

C#

```
Random rnd = new Random();
// Generate five random Boolean values.
for (int ctr = 1; ctr <= 5; ctr++) {
   bool bln = rnd.Next(0, 2) == 1;
   Console.WriteLine($"True or False: {bln}");
}

// The example displays an output similar to the following:
//       True or False: False
//       True or False: True
//       True or False: False
//       True or False: False
//       True or False: True
```

The following example calls the WriteLine(String, Object) method to display the current date. Note that the format item in the `format` argument uses the "D" standard date and time format string to display the date in the long date format of the current culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("Today's date: {0:D}", DateTime.Now);
   }
}
// The example displays output like the following:
//       Today's date: Monday, April 1, 2019
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The

resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{` *index*[`,` *alignment*][`:` *formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(UInt64)

> Important
>
> This API is not CLS-compliant.

Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the standard output stream.

C#

```
[System.CLSCompliant(false)]
public static void WriteLine (ulong value);
```

Parameters

`value` UInt64

The value to write.

Attributes CLSCompliantAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the

tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the UInt64.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()

- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(UInt32)

> Important
>
> This API is not CLS-compliant.

Writes the text representation of the specified 32-bit unsigned integer value, followed by the current line terminator, to the standard output stream.

C#

```
[System.CLSCompliant(false)]
public static void WriteLine (uint value);
```

Parameters

`value` UInt32

The value to write.

Attributes CLSCompliantAttribute

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the UInt32.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Single)

Writes the text representation of the specified single-precision floating-point value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (float value);
```

Parameters

`value` Single

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the Single.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Decimal)

Writes the text representation of the specified Decimal value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (decimal value);
```

Parameters

`value` Decimal

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the

tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the Decimal.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()

- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Int64)

Writes the text representation of the specified 64-bit signed integer value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (long value);
```

Parameters

`value` Int64

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the Int64.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Int32)

Writes the text representation of the specified 32-bit signed integer value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (int value);
```

Parameters

`value` Int32

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
```

```
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the Int32.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Double)

Writes the text representation of the specified double-precision floating-point value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (double value);
```

Parameters

`value` Double

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
```

```
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

The text representation of `value` is produced by calling the Double.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Char[])

Writes the specified array of Unicode characters, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (char[]? buffer);
```

Parameters

`buffer` Char[]

A Unicode character array.

Exceptions

IOException

An I/O error occurred.

Remarks

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Char)

Writes the specified Unicode character, followed by the current line terminator, value to the standard output stream.

C#

```
public static void WriteLine (char value);
```

Parameters

`value` Char

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
```

```
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine(("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23

Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Boolean)

Writes the text representation of the specified Boolean value, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (bool value);
```

Parameters

`value` Boolean

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example generates ten random integers and uses the Console.WriteLine(Boolean) method to indicate whether they are even.

C#

```
// Assign 10 random integers to an array.
Random rnd = new Random();
int[] numbers = new int[10];
for (int ctr = 0; ctr <= numbers.GetUpperBound(0); ctr++)
   numbers[ctr] = rnd.Next();

// Determine whether the numbers are even or odd.
foreach (var number in numbers) {
   bool even = (number % 2 == 0);
   Console.WriteLine("Is {0} even:", number);
   Console.WriteLine(even);
```

```
    Console.WriteLine();
}
```

Remarks

The text representation of `value` is produced by calling the Boolean.ToString method.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine()

Writes the current line terminator to the standard output stream.

C#

```
public static void WriteLine ();
```

Exceptions

IOException

An I/O error occurred.

Examples

The example changes the line terminator from its default value of "\r\n" or `vbCrLf` to "\r\n\r\n" or `vbCrLf` + `vbCrLf`. It then calls the WriteLine() and WriteLine(String) methods to display output to the console.

C#

```
string[] lines = { "This is the first line.",
                   "This is the second line." };
// Output the lines using the default newline sequence.
Console.WriteLine("With the default new line characters:");
Console.WriteLine();
foreach (string line in lines)
   Console.WriteLine(line);

Console.WriteLine();

// Redefine the newline characters to double space.
Console.Out.NewLine = "\r\n\r\n";
// Output the lines using the new newline sequence.
Console.WriteLine("With redefined new line characters:");
Console.WriteLine();
foreach (string line in lines)
   Console.WriteLine(line);

// The example displays the following output:
//       With the default new line characters:
//
//       This is the first line.
//       This is the second line.
//
//       With redefined new line characters:
//
//
//
//       This is the first line.
//
//       This is the second line.
```

Remarks

The default line terminator is a string whose value is a carriage return followed by a line feed ("\r\n" in C#, or `vbCrLf` in Visual Basic). You can change the line terminator by setting the TextWriter.NewLine property of the Out property to another string. The example provides an illustration.

See also

- Read()
- ReadLine()
- Write(String, Object)
- NewLine
- Out

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(String, Object, Object, Object)

Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.

C#

```
public static void WriteLine (string format, object? arg0, object? arg1,
object? arg2);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to write using `format`.

`arg1` Object

The second object to write using `format`.

`arg2` Object

The third object to write using `format`.

Exceptions

IOException

An I/O error occurred.

ArgumentNullException

`format` is `null`.

FormatException

The format specification in `format` is invalid.

Examples

The following example demonstrates the standard formatting specifiers for numbers, dates, and enumerations.

C#

```
// This code example demonstrates the Console.WriteLine() method.
// Formatting for this example uses the "en-US" culture.

using System;
class Sample
{
    enum Color {Yellow = 1, Blue, Green};
    static DateTime thisDate = DateTime.Now;

    public static void Main()
    {
        Console.Clear();

        // Format a negative integer or floating-point number in various
ways.
        Console.WriteLine("Standard Numeric Format Specifiers");
        Console.WriteLine(
            "(C) Currency: . . . . . . . . {0:C}\n" +
            "(D) Decimal:. . . . . . . . . {0:D}\n" +
            "(E) Scientific: . . . . . . . {1:E}\n" +
            "(F) Fixed point:. . . . . . . {1:F}\n" +
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(N) Number: . . . . . . . . . {0:N}\n" +
            "(P) Percent:. . . . . . . . . {1:P}\n" +
            "(R) Round-trip: . . . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
```

```
            -123, -123.45f);

        // Format the current date in various ways.
        Console.WriteLine("Standard DateTime Format Specifiers");
        Console.WriteLine(
            "(d) Short date: . . . . . . . {0:d}\n" +
            "(D) Long date:. . . . . . . . {0:D}\n" +
            "(t) Short time: . . . . . . . {0:t}\n" +
            "(T) Long time:. . . . . . . . {0:T}\n" +
            "(f) Full date/short time: . . {0:f}\n" +
            "(F) Full date/long time:. . . {0:F}\n" +
            "(g) General date/short time:. {0:g}\n" +
            "(G) General date/long time: . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(M) Month:. . . . . . . . . . {0:M}\n" +
            "(R) RFC1123:. . . . . . . . . {0:R}\n" +
            "(s) Sortable: . . . . . . . . {0:s}\n" +
            "(u) Universal sortable: . . . {0:u} (invariant)\n" +
            "(U) Universal full date/time: {0:U}\n" +
            "(Y) Year: . . . . . . . . . . {0:Y}\n",
            thisDate);

        // Format a Color enumeration value in various ways.
        Console.WriteLine("Standard Enumeration Format Specifiers");
        Console.WriteLine(
            "(G) General:. . . . . . . . . {0:G}\n" +
            "    (default):. . . . . . . . {0} (default = 'G')\n" +
            "(F) Flags:. . . . . . . . . . {0:F} (flags or integer)\n" +
            "(D) Decimal number: . . . . . {0:D}\n" +
            "(X) Hexadecimal:. . . . . . . {0:X}\n",
            Color.Green);
    }
}
/*
This code example produces the following results:

Standard Numeric Format Specifiers
(C) Currency: . . . . . . . . ($123.00)
(D) Decimal:. . . . . . . . . -123
(E) Scientific: . . . . . . . -1.234500E+002
(F) Fixed point:. . . . . . . -123.45
(G) General:. . . . . . . . . -123
    (default):. . . . . . . . -123 (default = 'G')
(N) Number: . . . . . . . . . -123.00
(P) Percent:. . . . . . . . . -12,345.00 %
(R) Round-trip: . . . . . . . -123.45
(X) Hexadecimal:. . . . . . . FFFFFF85

Standard DateTime Format Specifiers
(d) Short date: . . . . . . . 6/26/2004
(D) Long date:. . . . . . . . Saturday, June 26, 2004
(t) Short time: . . . . . . . 8:11 PM
(T) Long time:. . . . . . . . 8:11:04 PM
(f) Full date/short time: . . Saturday, June 26, 2004 8:11 PM
(F) Full date/long time:. . . Saturday, June 26, 2004 8:11:04 PM
```

```
(g) General date/short time:. 6/26/2004 8:11 PM
(G) General date/long time: . 6/26/2004 8:11:04 PM
    (default):. . . . . . . . 6/26/2004 8:11:04 PM (default = 'G')
(M) Month:. . . . . . . . . . June 26
(R) RFC1123:. . . . . . . . . Sat, 26 Jun 2004 20:11:04 GMT
(s) Sortable: . . . . . . . . 2004-06-26T20:11:04
(u) Universal sortable: . . . 2004-06-26 20:11:04Z (invariant)
(U) Universal full date/time: Sunday, June 27, 2004 3:11:04 AM
(Y) Year: . . . . . . . . . . June, 2004

Standard Enumeration Format Specifiers
(G) General:. . . . . . . . . Green
    (default):. . . . . . . . Green (default = 'G')
(F) Flags:. . . . . . . . . . Green (flags or integer)
(D) Decimal number: . . . . . 3
(X) Hexadecimal:. . . . . . . 00000003

*/
```

The following example is a tip calculator that calculates an 18% tip and uses the WriteLine method to display the amount of the original charge, the amount of the tip, and the total amount. The example is a console application that requires the user to supply the amount of the original charge as a command-line parameter.

C#

```
using System;

public class TipCalculator
{
    private const double tipRate = 0.18;
    public static void Main(string[] args)
    {
        double billTotal;
        if (args.Length == 0 || ! Double.TryParse(args[0], out
billTotal))
        {
            Console.WriteLine("usage: TIPCALC total");
            return;
        }
        double tip = billTotal * tipRate;
        Console.WriteLine();
        Console.WriteLine($"Bill total:\t{billTotal,8:c}");
        Console.WriteLine($"Tip total/rate:\t{tip,8:c} ({tipRate:p1})");
        Console.WriteLine("").PadRight(24, '-'));
        Console.WriteLine($"Grand total:\t{billTotal + tip,8:c}");
    }
}

/*
>tipcalc 52.23
```

```
Bill total:       $52.23
Tip total/rate:    $9.40 (18.0 %)
------------------------
Grand total:      $61.63
*/
```

Remarks

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string. The resulting string is written to the output stream.

The `format` parameter consists of zero or more runs of text intermixed with zero or more indexed placeholders, called format items, that correspond to an object in the parameter list of this method. The formatting process replaces each format item with the text representation of the value of the corresponding object.

The syntax of a format item is `{` *index*[`,` *alignment*][`:` *formatString*]`}`, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics.

- For more information about the composite formatting feature supported by methods such as Format, AppendFormat, and some overloads of WriteLine, see Composite Formatting.
- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about date and time format specifiers, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
- For more information about enumeration format specifiers, see Enumeration Format Strings.
- For more information about formatting, see Formatting Types.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)
- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

WriteLine(Object)

Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream.

C#

```
public static void WriteLine (object? value);
```

Parameters

`value` Object

The value to write.

Exceptions

IOException

An I/O error occurred.

Examples

The following example uses the WriteLine(Object) method to display each value in an object array to the console.

```
C#

Object[] values = { true, 12.632, 17908, "stringValue",
                            'a', 16907.32m };
foreach (var value in values)
   Console.WriteLine(value);

// The example displays the following output:
//    True
//    12.632
//    17908
//    stringValue
//    a
//    16907.32
```

Remarks

If `value` is `null`, only the line terminator is written. Otherwise, the `ToString` method of `value` is called to produce its string representation, and the resulting string is written to the standard output stream.

For more information about the line terminator, see the Remarks section of the WriteLine method that takes no parameters.

See also

- Read()
- ReadLine()
- Write(String, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Console.CancelKeyPress Event

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Occurs when the Control modifier key (Ctrl) and either the C console key (C) or the Break key are pressed simultaneously (Ctrl+C or Ctrl+Break).

C#

```
[System.Runtime.Versioning.UnsupportedOSPlatform("browser")]
[System.Runtime.Versioning.UnsupportedOSPlatform("android")]
[System.Runtime.Versioning.UnsupportedOSPlatform("ios")]
[System.Runtime.Versioning.UnsupportedOSPlatform("tvos")]
public static event ConsoleCancelEventHandler? CancelKeyPress;
```

Event Type

ConsoleCancelEventHandler

Attributes UnsupportedOSPlatformAttribute

Examples

The following example demonstrates how the CancelKeyPress event is used. When you press Ctrl+C, the read operation is interrupted and the `myHandler` event handler is invoked. Upon entry to the event handler, the ConsoleCancelEventArgs.Cancel property is `false`, which means that the current process will terminate when the event handler terminates. However, the event handler sets the ConsoleCancelEventArgs.Cancel property to `true`, which means that the process will not terminate and the read operation will resume.

C#

```
using System;

class Sample
{
```

```
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();

        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
        {
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");

            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
```

```
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
//    The read operation will resume...
//
//      Key pressed: Q
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: X
```

Remarks

This event is used in conjunction with System.ConsoleCancelEventHandler and System.ConsoleCancelEventArgs. The CancelKeyPress event enables a console application to intercept the Ctrl+C signal so the event handler can decide whether to continue executing or terminate. For more information about handling events, see Handling and Raising Events.

When the user presses either Ctrl+C or Ctrl+Break, the CancelKeyPress event is fired and the application's ConsoleCancelEventHandler event handler is executed. The event handler is passed a ConsoleCancelEventArgs object that has two useful properties:

- SpecialKey, which allows you to determine whether the handler was invoked as a result of the user pressing Ctrl+C (the property value is ConsoleSpecialKey.ControlC) or Ctrl+Break (the property value is ConsoleSpecialKey.ControlBreak).
- Cancel, which allows you to determine how your application should respond to the user pressing Ctrl+C or Ctrl+Break. By default, the Cancel property is `false`, which causes program execution to terminate when the event handler exits. Changing its property to `true` specifies that the application should continue to execute.

> **Tip**
>
> If your application has simple requirements, you can use the **TreatControlCAsInput** property instead of this event. By setting this property to `false`, you can ensure that your application always exits if the user presses Ctrl+C. By setting it to `true`, you can ensure that pressing Ctrl+C will not terminate the application.

The event handler for this event is executed on a thread pool thread.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ConsoleCancelEventArgs
- ConsoleCancelEventHandler
- TreatControlCAsInput

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleCancelEventArgs Class

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Provides data for the CancelKeyPress event. This class cannot be inherited.

C#

```
public sealed class ConsoleCancelEventArgs : EventArgs
```

Inheritance Object → EventArgs → ConsoleCancelEventArgs

Examples

The following example demonstrates how to use the ConsoleCancelEventArgs class to handle an event.

C#

```
using System;

class Sample
{
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();

        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
        {
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");
```

```
            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
//    The read operation will resume...
//
//      Key pressed: Q
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: X
```

Remarks

A user can interrupt a console application process by simultaneously pressing the Control modifier key and the C console key (Ctrl+C), or the Control modifier key and the Break console key (Ctrl+Break). Consequently, .NET provides a ConsoleCancelEventArgs object to the event handler for the Console.CancelKeyPress event to specify whether the process should be cancelled.

If the Cancel property is set to `true` in the event handler, the process is resumed; otherwise, the process is terminated. By default, the value of the ConsoleCancelEventArgs property is `false`, and the process terminates.

Properties

Expand table

Cancel	Gets or sets a value that indicates whether simultaneously pressing the Control modifier key and the C console key (Ctrl+C) or the Ctrl+Break keys terminates the current process. The default is `false`, which terminates the current process.
Special Key	Gets the combination of modifier and console keys that interrupted the current process.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ConsoleCancelEventArgs.Cancel Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets or sets a value that indicates whether simultaneously pressing the Control modifier key and the C console key (Ctrl+C) or the Ctrl+Break keys terminates the current process. The default is `false`, which terminates the current process.

```
C#

public bool Cancel { get; set; }
```

Property Value

Boolean

`true` if the current process should resume when the event handler concludes; `false` if the current process should terminate. The default value is `false`; the current process terminates when the event handler returns. If `true`, the current process continues.

Examples

The following example demonstrates how to use the Cancel property to handle an event.

```
C#

using System;

class Sample
{
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();
```

```
        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
        {
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");

            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
```

```
//    The read operation will resume...
//
//      Key pressed: Q
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: X
```

Remarks

The Cancel property is automatically initialized to `false` when the handler for the CancelKeyPress event is invoked. The value of the Cancel property when the event handler finishes determines whether the current process resumes or terminates.

In a set operation after Ctrl+C is pressed, specify `true` to indicate that the current process should resume when the event handler concludes, or `false` to indicate that the current process should terminate.

Notes to Callers

In the .NET Framework 3.5 and .NET Framework 4, attempting to set the Cancel property to `true` if the CancelKeyPress event was invoked by the user pressing Ctrl+Break threw an InvalidOperationException exception. In the .NET Framework 4.5, you can set the Cancel property to `true` after the user presses Ctrl+Break and cancel the termination of the application.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ConsoleCancelEventHandler
- ConsoleKey
- ConsoleModifiers

- SpecialKey

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleCancelEventArgs.SpecialKey Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the combination of modifier and console keys that interrupted the current process.

C#

```
public ConsoleSpecialKey SpecialKey { get; }
```

Property Value

ConsoleSpecialKey

One of the enumeration values that specifies the key combination that interrupted the current process. There is no default value.

Examples

The following example demonstrates how to use the SpecialKey property to handle an event.

C#

```
using System;

class Sample
{
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();

        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
        {
```

```
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");

            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
//    The read operation will resume...
//
//      Key pressed: Q
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
```

```
operation:
//      Key pressed: X
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleCancelEventHandler Delegate

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Represents the method that will handle the CancelKeyPress event of a Console.

C#

```
public delegate void ConsoleCancelEventHandler(object? sender,
ConsoleCancelEventArgs e);
```

Parameters

sender Object

The source of the event.

e ConsoleCancelEventArgs

A ConsoleCancelEventArgs object that contains the event data.

Examples

The following code example demonstrates how to use the ConsoleCancelEventHandler class to handle an event.

C#

```
using System;

class Sample
{
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();

        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
```

```
        {
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");

            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
//    The read operation will resume...
//
//      Key pressed: Q
//
```

```
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: X
```

Remarks

When you create a ConsoleCancelEventHandler delegate, you identify the method that will handle the event. To associate the event with your event handler, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate. For more information about event-handler delegates, see Handling and Raising Events.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleColor Enum

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Specifies constants that define foreground and background colors for the console.

C#

```
public enum ConsoleColor
```

Inheritance Object → ValueType → Enum → ConsoleColor

Fields

Expand table

Black	0	The color black.
Blue	9	The color blue.
Cyan	11	The color cyan (blue-green).
DarkBlue	1	The color dark blue.
DarkCyan	3	The color dark cyan (dark blue-green).
DarkGray	8	The color dark gray.
DarkGreen	2	The color dark green.
DarkMagenta	5	The color dark magenta (dark purplish-red).
DarkRed	4	The color dark red.
DarkYellow	6	The color dark yellow (ochre).
Gray	7	The color gray.
Green	10	The color green.
Magenta	13	The color magenta (purplish-red).

Red	12	The color red.
White	15	The color white.
Yellow	14	The color yellow.

Examples

The following example saves the values of the ConsoleColor enumeration to an array and stores the current values of the BackgroundColor and ForegroundColor properties to variables. It then changes the foreground color to each color in the ConsoleColor enumeration except to the color that matches the current background, and it changes the background color to each color in the ConsoleColor enumeration except to the color that matches the current foreground. (If the foreground color is the same as the background color, the text isn't visible.) Finally, it calls the ResetColor method to restore the original console colors.

C#

```
using System;

class Example
{
   public static void Main()
   {
      // Get an array with the values of ConsoleColor enumeration members.
      ConsoleColor[] colors = (ConsoleColor[])
ConsoleColor.GetValues(typeof(ConsoleColor));
      // Save the current background and foreground colors.
      ConsoleColor currentBackground = Console.BackgroundColor;
      ConsoleColor currentForeground = Console.ForegroundColor;

      // Display all foreground colors except the one that matches the
background.
      Console.WriteLine("All the foreground colors except {0}, the
background color:",
                        currentBackground);
      foreach (var color in colors) {
         if (color == currentBackground) continue;

         Console.ForegroundColor = color;
         Console.WriteLine("   The foreground color is {0}.", color);
      }
      Console.WriteLine();
      // Restore the foreground color.
      Console.ForegroundColor = currentForeground;

      // Display each background color except the one that matches the
current foreground color.
```

```
      Console.WriteLine("All the background colors except {0}, the
foreground color:",
                        currentForeground);
      foreach (var color in colors) {
         if (color == currentForeground) continue;

         Console.BackgroundColor = color;
         Console.WriteLine("   The background color is {0}.", color);
      }

      // Restore the original console colors.
      Console.ResetColor();
      Console.WriteLine("\nOriginal colors restored...");
   }
}
//The example displays output like the following:
//    All the foreground colors except DarkCyan, the background color:
//       The foreground color is Black.
//       The foreground color is DarkBlue.
//       The foreground color is DarkGreen.
//       The foreground color is DarkRed.
//       The foreground color is DarkMagenta.
//       The foreground color is DarkYellow.
//       The foreground color is Gray.
//       The foreground color is DarkGray.
//       The foreground color is Blue.
//       The foreground color is Green.
//       The foreground color is Cyan.
//       The foreground color is Red.
//       The foreground color is Magenta.
//       The foreground color is Yellow.
//       The foreground color is White.
//
//    All the background colors except White, the foreground color:
//       The background color is Black.
//       The background color is DarkBlue.
//       The background color is DarkGreen.
//       The background color is DarkCyan.
//       The background color is DarkRed.
//       The background color is DarkMagenta.
//       The background color is DarkYellow.
//       The background color is Gray.
//       The background color is DarkGray.
//       The background color is Blue.
//       The background color is Green.
//       The background color is Cyan.
//       The background color is Red.
//       The background color is Magenta.
//       The background color is Yellow.
//
//    Original colors restored...
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- Console

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKey Enum

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Specifies the standard keys on a console.

C#

```
public enum ConsoleKey
```

Inheritance Object → ValueType → Enum → ConsoleKey

Fields

Expand table

A	65	The A key.
Add	107	The Add key (the addition key on the numeric keypad).
Applications	93	The Application key (Microsoft Natural Keyboard).
Attention	246	The ATTN key.
B	66	The B key.
Backspace	8	The BACKSPACE key.
BrowserBack	166	The Browser Back key.
BrowserFavorites	171	The Browser Favorites key.
BrowserForward	167	The Browser Forward key.
BrowserHome	172	The Browser Home key.
BrowserRefresh	168	The Browser Refresh key.
BrowserSearch	170	The Browser Search key.
BrowserStop	169	The Browser Stop key.

C	67	The C key.
Clear	12	The CLEAR key.
CrSel	247	The CRSEL (CURSOR SELECT) key.
D	68	The D key.
D0	48	The 0 key.
D1	49	The 1 key.
D2	50	The 2 key.
D3	51	The 3 key.
D4	52	The 4 key.
D5	53	The 5 key.
D6	54	The 6 key.
D7	55	The 7 key.
D8	56	The 8 key.
D9	57	The 9 key.
Decimal	110	The Decimal key (the decimal key on the numeric keypad).
Delete	46	The DEL (DELETE) key.
Divide	111	The Divide key (the division key on the numeric keypad).
DownArrow	40	The DOWN ARROW key.
E	69	The E key.
End	35	The END key.
Enter	13	The ENTER key.
EraseEndOfFile	249	The ERASE EOF key.
Escape	27	The ESC (ESCAPE) key.
Execute	43	The EXECUTE key.
ExSel	248	The EXSEL (EXTEND SELECTION) key.
F	70	The F key.
F1	112	The F1 key.

F10	121	The F10 key.
F11	122	The F11 key.
F12	123	The F12 key.
F13	124	The F13 key.
F14	125	The F14 key.
F15	126	The F15 key.
F16	127	The F16 key.
F17	128	The F17 key.
F18	129	The F18 key.
F19	130	The F19 key.
F2	113	The F2 key.
F20	131	The F20 key.
F21	132	The F21 key.
F22	133	The F22 key.
F23	134	The F23 key.
F24	135	The F24 key.
F3	114	The F3 key.
F4	115	The F4 key.
F5	116	The F5 key.
F6	117	The F6 key.
F7	118	The F7 key.
F8	119	The F8 key.
F9	120	The F9 key.
G	71	The G key.
H	72	The H key.
Help	47	The HELP key.
Home	36	The HOME key.

I	73	The I key.
Insert	45	The INS (INSERT) key.
J	74	The J key.
K	75	The K key.
L	76	The L key.
LaunchApp1	182	The Start Application 1 key (Microsoft Natural Keyboard).
LaunchApp2	183	The Start Application 2 key (Microsoft Natural Keyboard).
LaunchMail	180	The Start Mail key (Microsoft Natural Keyboard).
LaunchMediaSelect	181	The Select Media key (Microsoft Natural Keyboard).
LeftArrow	37	The LEFT ARROW key.
LeftWindows	91	The left Windows logo key (Microsoft Natural Keyboard).
M	77	The M key.
MediaNext	176	The Media Next Track key.
MediaPlay	179	The Media Play/Pause key.
MediaPrevious	177	The Media Previous Track key.
MediaStop	178	The Media Stop key.
Multiply	106	The Multiply key (the multiplication key on the numeric keypad).
N	78	The N key.
NoName	252	A constant reserved for future use.
None	0	
NumPad0	96	The 0 key on the numeric keypad.
NumPad1	97	The 1 key on the numeric keypad.
NumPad2	98	The 2 key on the numeric keypad.
NumPad3	99	The 3 key on the numeric keypad.
NumPad4	100	The 4 key on the numeric keypad.
NumPad5	101	The 5 key on the numeric keypad.
NumPad6	102	The 6 key on the numeric keypad.

NumPad7	103	The 7 key on the numeric keypad.
NumPad8	104	The 8 key on the numeric keypad.
NumPad9	105	The 9 key on the numeric keypad.
O	79	The O key.
Oem1	186	The OEM 1 key (OEM specific).
Oem102	226	The OEM 102 key (OEM specific).
Oem2	191	The OEM 2 key (OEM specific).
Oem3	192	The OEM 3 key (OEM specific).
Oem4	219	The OEM 4 key (OEM specific).
Oem5	220	The OEM 5 (OEM specific).
Oem6	221	The OEM 6 key (OEM specific).
Oem7	222	The OEM 7 key (OEM specific).
Oem8	223	The OEM 8 key (OEM specific).
OemClear	254	The CLEAR key (OEM specific).
OemComma	188	The OEM Comma key on any country/region keyboard.
OemMinus	189	The OEM Minus key on any country/region keyboard.
OemPeriod	190	The OEM Period key on any country/region keyboard.
OemPlus	187	The OEM Plus key on any country/region keyboard.
P	80	The P key.
Pa1	253	The PA1 key.
Packet	231	The PACKET key (used to pass Unicode characters with keystrokes).
PageDown	34	The PAGE DOWN key.
PageUp	33	The PAGE UP key.
Pause	19	The PAUSE key.
Play	250	The PLAY key.
Print	42	The PRINT key.
PrintScreen	44	The PRINT SCREEN key.

Process	229	The IME PROCESS key.
Q	81	The Q key.
R	82	The R key.
RightArrow	39	The RIGHT ARROW key.
RightWindows	92	The right Windows logo key (Microsoft Natural Keyboard).
S	83	The S key.
Select	41	The SELECT key.
Separator	108	The Separator key.
Sleep	95	The Computer Sleep key.
Spacebar	32	The SPACEBAR key.
Subtract	109	The Subtract key (the subtraction key on the numeric keypad).
T	84	The T key.
Tab	9	The TAB key.
U	85	The U key.
UpArrow	38	The UP ARROW key.
V	86	The V key.
VolumeDown	174	The Volume Down key (Microsoft Natural Keyboard).
VolumeMute	173	The Volume Mute key (Microsoft Natural Keyboard).
VolumeUp	175	The Volume Up key (Microsoft Natural Keyboard).
W	87	The W key.
X	88	The X key.
Y	89	The Y key.
Z	90	The Z key.
Zoom	251	The ZOOM key.

Examples

The following example uses the ConsoleKey enumeration to indicate to the user which key the user had pressed.

C#

```
using System;
using System.Text;

public class ConsoleKeyExample
{
   public static void Main()
   {
      ConsoleKeyInfo input;
      do {
         Console.WriteLine("Press a key, together with Alt, Ctrl, or
Shift.");
         Console.WriteLine("Press Esc to exit.");
         input = Console.ReadKey(true);

         StringBuilder output = new StringBuilder(
                        String.Format("You pressed {0}",
input.Key.ToString()));
         bool modifiers = false;

         if (input.Modifiers.HasFlag(ConsoleModifiers.Alt)) {
            output.Append(", together with " +
ConsoleModifiers.Alt.ToString());
            modifiers = true;
         }
         if (input.Modifiers.HasFlag(ConsoleModifiers.Control))
         {
            if (modifiers) {
               output.Append(" and ");
            }
            else {
               output.Append(", together with ");
               modifiers = true;
            }
            output.Append(ConsoleModifiers.Control.ToString());
         }
         if (input.Modifiers.HasFlag(ConsoleModifiers.Shift))
         {
            if (modifiers) {
               output.Append(" and ");
            }
            else {
               output.Append(", together with ");
               modifiers = true;
            }
            output.Append(ConsoleModifiers.Shift.ToString());
         }
         output.Append(".");
         Console.WriteLine(output.ToString());
         Console.WriteLine();
```

```
        } while (input.Key != ConsoleKey.Escape);
    }
}
// The output from a sample console session might appear as follows:
//       Press a key, together with Alt, Ctrl, or Shift.
//       Press Esc to exit.
//       You pressed D.
//
//       Press a key, together with Alt, Ctrl, or Shift.
//       Press Esc to exit.
//       You pressed X, together with Shift.
//
//       Press a key, together with Alt, Ctrl, or Shift.
//       Press Esc to exit.
//       You pressed L, together with Control and Shift.
//
//       Press a key, together with Alt, Ctrl, or Shift.
//       Press Esc to exit.
//       You pressed P, together with Alt and Control and Shift.
//
//       Press a key, together with Alt, Ctrl, or Shift.
//       Press Esc to exit.
//       You pressed Escape.
```

Remarks

The ConsoleKey enumeration is typically used in the nfo structure, which is returned by the Console.ReadKey method to indicate which key on the console has been pressed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ConsoleKeyInfo Struct

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Describes the console key that was pressed, including the character represented by the console key and the state of the SHIFT, ALT, and CTRL modifier keys.

C#

```
public readonly struct ConsoleKeyInfo : IEquatable<ConsoleKeyInfo>
```

Inheritance Object → ValueType → ConsoleKeyInfo

Implements IEquatable<ConsoleKeyInfo>

Examples

The following example demonstrates using a ConsoleKeyInfo object in a read operation.

C#

```
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT, and a
console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
         if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
```

```
            if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
            if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
            Console.WriteLine(cki.Key.ToString());
        } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Remarks

The ConsoleKeyInfo type is not intended to be created by users. Instead, it is returned to the user in response to calling the Console.ReadKey method.

The ConsoleKeyInfo object describes the ConsoleKey constant and Unicode character, if any, that correspond to the pressed console key. The ConsoleKeyInfo object also describes, in a bitwise combination of ConsoleModifiers values, whether one or more SHIFT, ALT, or CTRL modifier keys was pressed simultaneously with the console key.

Constructors

Expand table

ConsoleKeyInfo(Char, ConsoleKey, Boolean, Boolean, Boolean)	Initializes a new instance of the ConsoleKeyInfo structure using the specified character, console key, and modifier keys.

Properties

Expand table

Key	Gets the console key represented by the current ConsoleKeyInfo object.

KeyChar	Gets the Unicode character represented by the current ConsoleKeyInfo object.
Modifiers	Gets a bitwise combination of ConsoleModifiers values that specifies one or more modifier keys pressed simultaneously with the console key.

Methods

Expand table

Equals(ConsoleKeyInfo)	Gets a value indicating whether the specified ConsoleKeyInfo object is equal to the current ConsoleKeyInfo object.
Equals(Object)	Gets a value indicating whether the specified object is equal to the current ConsoleKeyInfo object.
GetHashCode()	Returns the hash code for the current ConsoleKeyInfo object.

Operators

Expand table

Equality(ConsoleKeyInfo, ConsoleKeyInfo)	Indicates whether the specified ConsoleKeyInfo objects are equal.
Inequality(ConsoleKeyInfo, ConsoleKeyInfo)	Indicates whether the specified ConsoleKeyInfo objects are not equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ConsoleModifiers
- ConsoleKey

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ConsoleKeyInfo(Char, ConsoleKey, Boolean, Boolean, Boolean) Constructor

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Initializes a new instance of the ConsoleKeyInfo structure using the specified character, console key, and modifier keys.

C#

```
public ConsoleKeyInfo (char keyChar, ConsoleKey key, bool shift, bool alt,
bool control);
```

Parameters

keyChar Char

The Unicode character that corresponds to the `key` parameter.

key ConsoleKey

The console key that corresponds to the `keyChar` parameter.

shift Boolean

`true` to indicate that a SHIFT key was pressed; otherwise, `false`.

alt Boolean

`true` to indicate that an ALT key was pressed; otherwise, `false`.

control Boolean

`true` to indicate that a CTRL key was pressed; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

The numeric value of the `key` parameter is less than 0 or greater than 255.

Remarks

This type is not intended to be created by users. Instead, it is returned to the user in response to calling the Console.ReadKey method.

The ConsoleKeyInfo type does not specify whether the left or right SHIFT, ALT, or CTRL modifier key was pressed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKeyInfo.Key Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the console key represented by the current ConsoleKeyInfo object.

C#

```
public ConsoleKey Key { get; }
```

Property Value

ConsoleKey

A value that identifies the console key that was pressed.

Examples

The following example demonstrates using a Key property in a read operation.

C#

```
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT, and a
console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
         if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
```

```
            if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
            if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
            Console.WriteLine(cki.Key.ToString());
        } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKeyInfo.KeyChar Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets the Unicode character represented by the current ConsoleKeyInfo object.

C#

```
public char KeyChar { get; }
```

Property Value

Char

An object that corresponds to the console key represented by the current ConsoleKeyInfo object.

Examples

The following example uses the KeyChar property to add the characters input by the user into a string. The example ignores special keys other than **ENTER**, **ESC**, and **BACKSPACE**.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Configure console.
      Console.BufferWidth = 80;
      Console.WindowWidth = Console.BufferWidth;
      Console.TreatControlCAsInput = true;

      string inputString = String.Empty;
      ConsoleKeyInfo keyInfo;

      Console.WriteLine("Enter a string. Press <Enter> or Esc to exit.");
```

```
      do {
         keyInfo = Console.ReadKey(true);
         // Ignore if Alt or Ctrl is pressed.
         if ((keyInfo.Modifiers & ConsoleModifiers.Alt) ==
ConsoleModifiers.Alt)
            continue;
         if ((keyInfo.Modifiers & ConsoleModifiers.Control) ==
ConsoleModifiers.Control)
            continue;
         // Ignore if KeyChar value is \u0000.
         if (keyInfo.KeyChar == '\u0000') continue;
         // Ignore tab key.
         if (keyInfo.Key == ConsoleKey.Tab) continue;
         // Handle backspace.
         if (keyInfo.Key == ConsoleKey.Backspace) {
            // Are there any characters to erase?
            if (inputString.Length >= 1) {
               // Determine where we are in the console buffer.
               int cursorCol = Console.CursorLeft - 1;
               int oldLength = inputString.Length;
               int extraRows = oldLength / 80;

               inputString = inputString.Substring(0, oldLength - 1);
               Console.CursorLeft = 0;
               Console.CursorTop = Console.CursorTop - extraRows;
               Console.Write(inputString + new String(' ', oldLength -
inputString.Length));
               Console.CursorLeft = cursorCol;
            }
            continue;
         }
         // Handle Escape key.
         if (keyInfo.Key == ConsoleKey.Escape) break;
         // Handle key by adding it to input string.
         Console.Write(keyInfo.KeyChar);
         inputString += keyInfo.KeyChar;
      } while (keyInfo.Key != ConsoleKey.Enter);
      Console.WriteLine("\n\nYou entered:\n    {0}",
                        String.IsNullOrEmpty(inputString) ? "<nothing>" :
inputString);
   }
}
```

Remarks

If the key pressed does not map to a Unicode character (for example, if the user presses the **F1** key or the **Home** key), the value of the KeyChar property is \U0000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ConsoleKeyInfo.Modifiers Property

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a bitwise combination of ConsoleModifiers values that specifies one or more modifier keys pressed simultaneously with the console key.

C#

```
public ConsoleModifiers Modifiers { get; }
```

Property Value

ConsoleModifiers

A bitwise combination of the enumeration values. There is no default value.

Examples

The following example demonstrates using a Modifiers property in a read operation.

C#

```
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT, and a
console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
```

```
            if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
            if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
            if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
            Console.WriteLine(cki.Key.ToString());
        } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKeyInfo.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Gets a value indicating whether an object or specified ConsoleKeyInfo object is equal to the current ConsoleKeyInfo object.

Overloads

Expand table

Equals(ConsoleKeyInfo)	Gets a value indicating whether the specified ConsoleKeyInfo object is equal to the current ConsoleKeyInfo object.
Equals(Object)	Gets a value indicating whether the specified object is equal to the current ConsoleKeyInfo object.

Equals(ConsoleKeyInfo)

Gets a value indicating whether the specified ConsoleKeyInfo object is equal to the current ConsoleKeyInfo object.

C#

```
public bool Equals (ConsoleKeyInfo obj);
```

Parameters

obj ConsoleKeyInfo

An object to compare to the current ConsoleKeyInfo object.

Returns

Boolean

`true` if `obj` is equal to the current ConsoleKeyInfo object; otherwise, `false`.

Implements

Equals(T)

Remarks

Two ConsoleKeyInfo objects are equal if their corresponding KeyChar, Key, and Modifiers properties are equal.

The Equals method performs slightly better than the ConsoleKeyInfo.Equals(Object) method because it does not have to convert `obj` to an object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Equals(Object)

Gets a value indicating whether the specified object is equal to the current ConsoleKeyInfo object.

C#

```
public override bool Equals (object? value);
```

Parameters

`value` Object

An object to compare to the current ConsoleKeyInfo object.

Returns

Boolean

`true` if `value` is a ConsoleKeyInfo object and is equal to the current ConsoleKeyInfo object; otherwise, `false`.

Examples

The following example demonstrates the Equals method.

C#

```
// This example demonstrates the ConsoleKeyInfo.Equals() method.

using System;
using System.Text;

class Sample
{
    public static void Main()
    {
    string k1 = "\nEnter a key ......... ";
    string k2 = "\nEnter another key ... ";
    string key1 = "";
    string key2 = "";
    string areKeysEqual = "The {0} and {1} keys are {2}equal.";
    string equalValue = "";
    string prompt = "Press the escape key (ESC) to quit, " +
                    "or any other key to continue.";
    ConsoleKeyInfo cki1;
    ConsoleKeyInfo cki2;

//
// The Console.TreatControlCAsInput property prevents this example from
// ending if you press CTL+C, however all other operating system keys and
// shortcuts, such as ALT+TAB or the Windows Logo key, are still in
effect.
//
    Console.TreatControlCAsInput = true;

// Request that the user enter two key presses. A key press and any
// combination shift, CTRL, and ALT modifier keys is permitted.
    do
    {
        Console.Write(k1);
        cki1 = Console.ReadKey(false);
        Console.Write(k2);
        cki2 = Console.ReadKey(false);
        Console.WriteLine();
//
        key1 = KeyCombination(cki1);
```

```
        key2 = KeyCombination(cki2);
        if (cki1.Equals(cki2))
            equalValue = "";
        else
            equalValue = "not ";
        Console.WriteLine(areKeysEqual, key1, key2, equalValue);
//
        Console.WriteLine(prompt);
        cki1 = Console.ReadKey(true);
    } while (cki1.Key != ConsoleKey.Escape);
// Note: This example requires the Escape (Esc) key.
    }

// The KeyCombination() method creates a string that specifies what
// key and what combination of shift, CTRL, and ALT modifier keys
// were pressed simultaneously.

    protected static string KeyCombination(ConsoleKeyInfo sourceCki)
    {
    StringBuilder sb = new StringBuilder();
    sb.Length = 0;
    string keyCombo;
    if (sourceCki.Modifiers != 0)
        {
        if ((sourceCki.Modifiers & ConsoleModifiers.Alt) != 0)
            sb.Append("ALT+");
        if ((sourceCki.Modifiers & ConsoleModifiers.Shift) != 0)
            sb.Append("SHIFT+");
        if ((sourceCki.Modifiers & ConsoleModifiers.Control) != 0)
            sb.Append("CTL+");
        }
    sb.Append(sourceCki.Key.ToString());
    keyCombo = sb.ToString();
    return keyCombo;
    }
}

/*
This example produces results similar to the following output:

Enter a key ......... a
Enter another key ... a
The A and A keys are equal.
Press the escape key (ESC) to quit, or any other key to continue.

Enter a key ......... a
Enter another key ... A
The A and SHIFT+A keys are not equal.
Press the escape key (ESC) to quit, or any other key to continue.

Enter a key ......... S
Enter another key ...
The ALT+SHIFT+S and ALT+CTL+F keys are not equal.
Press the escape key (ESC) to quit, or any other key to continue.
```

```
Enter a key .........
Enter another key ...
The UpArrow and UpArrow keys are equal.
Press the escape key (ESC) to quit, or any other key to continue.

*/
```

Remarks

Two ConsoleKeyInfo objects are equal if their corresponding KeyChar, Key, and Modifiers properties are equal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKeyInfo.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Returns the hash code for the current ConsoleKeyInfo object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Examples

The following example demonstrates the GetHashCode method.

C#

```
// This example demonstrates the ConsoleKeyInfo.GetHashCode() method.

using System;
using System.Text;

class Sample
{
    public static void Main()
    {
    string k1 = "\nEnter a key ......... ";
    string key1 = "";
    string hashCodeFmt = "The hash code for the {0} key is {1}.";
    string prompt = "Press the escape key (ESC) to quit, " +
                    "or any other key to continue.";
    ConsoleKeyInfo cki1;
    int hashCode = 0;

//
// The Console.TreatControlCAsInput property prevents this example from
```

```
// ending if you press CTL+C, however all other operating system keys and
// shortcuts, such as ALT+TAB or the Windows Logo key, are still in effect.
//
    Console.TreatControlCAsInput = true;

// Request that the user enter two key presses. A key press and any
// combination shift, CTRL, and ALT modifier keys is permitted.
    do
    {
        Console.Write(k1);
        cki1 = Console.ReadKey(false);
        Console.WriteLine();
//
        key1 = KeyCombination(cki1);
        hashCode = cki1.GetHashCode();
        Console.WriteLine(hashCodeFmt, key1, hashCode);
//
        Console.WriteLine(prompt);
        cki1 = Console.ReadKey(true);
    } while (cki1.Key != ConsoleKey.Escape);
// Note: This example requires the Escape (Esc) key.
    }

// The KeyCombination() method creates a string that specifies what
// key and what combination of shift, CTRL, and ALT modifier keys
// were pressed simultaneously.

    protected static string KeyCombination(ConsoleKeyInfo sourceCki)
    {
    StringBuilder sb = new StringBuilder();
    sb.Length = 0;
    string keyCombo;
    if (sourceCki.Modifiers != 0)
        {
        if ((sourceCki.Modifiers & ConsoleModifiers.Alt) != 0)
            sb.Append("ALT+");
        if ((sourceCki.Modifiers & ConsoleModifiers.Shift) != 0)
            sb.Append("SHIFT+");
        if ((sourceCki.Modifiers & ConsoleModifiers.Control) != 0)
            sb.Append("CTL+");
        }
    sb.Append(sourceCki.Key.ToString());
    keyCombo = sb.ToString();
    return keyCombo;
    }
}

/*
This example produces results similar to the following output:

Enter a key ......... a
The hash code for the A key is 97.
Press the escape key (ESC) to quit, or any other key to continue.

Enter a key ......... S
```

```
The hash code for the SHIFT+S key is 83.
Press the escape key (ESC) to quit, or any other key to continue.

Enter a key .........
The hash code for the ALT+SHIFT+CTL+J key is 7.
Press the escape key (ESC) to quit, or any other key to continue.

*/
```

Remarks

The value returned by the GetHashCode method is not suitable for distinguishing one ConsoleKeyInfo object from another. If your application needs a unique hash code, override the GetHashCode method with your own method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleKeyInfo.Equality(ConsoleKeyInfo, ConsoleKeyInfo) Operator

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Indicates whether the specified ConsoleKeyInfo objects are equal.

C#

```
public static bool operator == (ConsoleKeyInfo a, ConsoleKeyInfo b);
```

Parameters

a ConsoleKeyInfo

The first object to compare.

b ConsoleKeyInfo

The second object to compare.

Returns

Boolean

`true` if `a` is equal to `b`; otherwise, `false`.

Remarks

Two ConsoleKeyInfo objects are equal if their corresponding KeyChar, Key, and Modifiers properties are equal.

The equivalent method for this operator is ConsoleKeyInfo.Equals(ConsoleKeyInfo).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ConsoleKeyInfo.Inequality(ConsoleKeyInfo, ConsoleKeyInfo) Operator

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Indicates whether the specified ConsoleKeyInfo objects are not equal.

```
C#
public static bool operator != (ConsoleKeyInfo a, ConsoleKeyInfo b);
```

Parameters

a ConsoleKeyInfo

The first object to compare.

b ConsoleKeyInfo

The second object to compare.

Returns

Boolean

`true` if `a` is not equal to `b`; otherwise, `false`.

Remarks

Two ConsoleKeyInfo objects are equal if their corresponding KeyChar, Key, and Modifiers properties are equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ConsoleModifiers Enum

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Represents the SHIFT, ALT, and CTRL modifier keys on a keyboard.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum ConsoleModifiers
```

Inheritance Object → ValueType → Enum → ConsoleModifiers

Attributes FlagsAttribute

Fields

Expand table

Alt	1	The left or right ALT modifier key.
Control	4	The left or right CTRL modifier key.
None	0	
Shift	2	The left or right SHIFT modifier key.

Examples

The following code example reads a key and determines whether one or more modifier keys was pressed.

C#

```
using System;

class Example
{
   public static void Main()
   {
      ConsoleKeyInfo cki;
      // Prevent example from ending if CTL+C is pressed.
      Console.TreatControlCAsInput = true;

      Console.WriteLine("Press any combination of CTL, ALT, and SHIFT, and a
console key.");
      Console.WriteLine("Press the Escape (Esc) key to quit: \n");
      do
      {
         cki = Console.ReadKey();
         Console.Write(" --- You pressed ");
         if((cki.Modifiers & ConsoleModifiers.Alt) != 0)
Console.Write("ALT+");
         if((cki.Modifiers & ConsoleModifiers.Shift) != 0)
Console.Write("SHIFT+");
         if((cki.Modifiers & ConsoleModifiers.Control) != 0)
Console.Write("CTL+");
         Console.WriteLine(cki.Key.ToString());
       } while (cki.Key != ConsoleKey.Escape);
    }
}
// This example displays output similar to the following:
//       Press any combination of CTL, ALT, and SHIFT, and a console key.
//       Press the Escape (Esc) key to quit:
//
//       a --- You pressed A
//       k --- You pressed ALT+K
//       ► --- You pressed CTL+P
//         --- You pressed RightArrow
//       R --- You pressed SHIFT+R
//                --- You pressed CTL+I
//       j --- You pressed ALT+J
//       O --- You pressed SHIFT+O
//       § --- You pressed CTL+U
```

Remarks

No distinction is made between the left or right SHIFT, ALT, and CTRL keys.

The ConsoleModifiers enumeration is used in conjunction with the ConsoleKeyInfo type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ConsoleKeyInfo
- ReadKey

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ConsoleSpecialKey Enum

Reference

Definition

Namespace: System

Assembly: System.Console.dll

Specifies combinations of modifier and console keys that can interrupt the current process.

C#

```
public enum ConsoleSpecialKey
```

Inheritance Object → ValueType → Enum → ConsoleSpecialKey

Fields

Expand table

ControlBreak	1	The Control modifier key plus the BREAK console key.
ControlC	0	The Control modifier key plus the C console key.

Examples

The following code example displays the ConsoleSpecialKey value that invokes the associated event handler.

C#

```
using System;

class Sample
{
    public static void Main()
    {
        ConsoleKeyInfo cki;

        Console.Clear();
```

```
        // Establish an event handler to process key press events.
        Console.CancelKeyPress += new ConsoleCancelEventHandler(myHandler);
        while (true)
        {
            Console.Write("Press any key, or 'X' to quit, or ");
            Console.WriteLine("CTRL+C to interrupt the read operation:");

            // Start a console read operation. Do not display the input.
            cki = Console.ReadKey(true);

            // Announce the name of the key that was pressed .
            Console.WriteLine($"  Key pressed: {cki.Key}\n");

            // Exit if the user pressed the 'X' key.
            if (cki.Key == ConsoleKey.X) break;
        }
    }

    protected static void myHandler(object sender, ConsoleCancelEventArgs
args)
    {
        Console.WriteLine("\nThe read operation has been interrupted.");

        Console.WriteLine($"  Key pressed: {args.SpecialKey}");

        Console.WriteLine($"  Cancel property: {args.Cancel}");

        // Set the Cancel property to true to prevent the process from
terminating.
        Console.WriteLine("Setting the Cancel property to true...");
        args.Cancel = true;

        // Announce the new value of the Cancel property.
        Console.WriteLine($"  Cancel property: {args.Cancel}");
        Console.WriteLine("The read operation will resume...\n");
    }
}
// The example displays output similar to the following:
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: J
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: Enter
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//
//    The read operation has been interrupted.
//      Key pressed: ControlC
//      Cancel property: False
//    Setting the Cancel property to true...
//      Cancel property: True
//    The read operation will resume...
```

```
//
//      Key pressed: Q
//
//    Press any key, or 'X' to quit, or CTRL+C to interrupt the read
operation:
//      Key pressed: X
```

Remarks

The ConsoleCancelEventArgs.SpecialKey property returns the ConsoleSpecialKey enumeration type.

Simultaneously pressing certain key combinations signals the operating system to interrupt the currently running process. The two valid key combinations are ConsoleModifiers.Control plus BREAK (CTRL+BREAK) and ConsoleModifiers.Control plus ConsoleKey.C (CTRL+C).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.6, 2.0, 2.1

See also

- ConsoleCancelEventArgs

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ContextBoundObject Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines the base class for all context-bound classes.

C#

```
public abstract class ContextBoundObject : MarshalByRefObject
```

Inheritance Object → MarshalByRefObject → ContextBoundObject

Derived System.EnterpriseServices.ServicedComponent
System.ServiceModel.ComIntegration.ServiceMoniker

Remarks

Objects that reside in a context and are bound to the context rules are called context-bound objects. A context is a set of properties or usage rules that define an environment where a collection of objects resides. The rules are enforced when the objects are entering or leaving a context. Objects that are not context-bound are called agile objects.

Contexts are created during object activation. A new object is placed into an existing context or into a new context created using the attributes included in the metadata of the type. Context-bound classes are marked with a ContextAttribute that provides the usage rules. The context properties that can be added include policies regarding synchronization and transactions.

Notes to Implementers

The current version of the common language runtime does not support generic ContextBoundObject types or nongeneric ContextBoundObject types that have generic methods. Attempting to create an instance of such a type causes a TypeLoadException.

Constructors

Expand table

ContextBoundObject()	Instantiates an instance of the ContextBoundObject class.

Methods

Expand table

CreateObjRef(Type)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (Inherited from MarshalByRefObject)
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetLifetimeService()	**Obsolete.** Retrieves the current lifetime service object that controls the lifetime policy for this instance. (Inherited from MarshalByRefObject)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeLifetime Service()	**Obsolete.** Obtains a lifetime service object to control the lifetime policy for this instance. (Inherited from MarshalByRefObject)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Memberwise Clone(Boolean)	Creates a shallow copy of the current MarshalByRefObject object. (Inherited from MarshalByRefObject)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SynchronizationAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ContextBoundObject Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Instantiates an instance of the ContextBoundObject class.

C#

```
protected ContextBoundObject ();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ContextBoundObject

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ContextMarshalException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt to marshal an object across a context boundary fails.

C#

```
public class ContextMarshalException : SystemException
```

Inheritance Object → Exception → SystemException → ContextMarshalException

Remarks

Objects can marshal by value or by reference. Any attempt to pass a type that cannot be marshaled through a context boundary will result in a ContextMarshalException.

ContextMarshalException uses the HRESULT COR_E_CONTEXTMARSHAL which has the value 0x80131504L.

ContextMarshalException uses the default Equals implementation, which supports reference equality.

For a list of initial property values for an instance of ContextMarshalException, see the ContextMarshalException constructors.

Constructors

Expand table

ContextMarshalException()	Initializes a new instance of the ContextMarshalException class with default properties.
ContextMarshalException(SerializationInfo, StreamingContext)	**Obsolete.**

	Initializes a new instance of the ContextMarshalException class with serialized data.
ContextMarshalException(String)	Initializes a new instance of the ContextMarshalException class with a specified error message.
ContextMarshalException(String, Exception)	Initializes a new instance of the ContextMarshalException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.

	(Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1 *(2.0, 3.0, 3.5)*
.NET Standard	2.0, 2.1

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ContextMarshalException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ContextMarshalException class.

Overloads

Expand table

ContextMarshalException()	Initializes a new instance of the ContextMarshalException class with default properties.
ContextMarshalException(String)	Initializes a new instance of the ContextMarshalException class with a specified error message.
ContextMarshalException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ContextMarshalException class with serialized data.
ContextMarshalException(String, Exception)	Initializes a new instance of the ContextMarshalException class with a specified error message and a reference to the inner exception that is the cause of this exception.

ContextMarshalException()

Initializes a new instance of the ContextMarshalException class with default properties.

C#

```
public ContextMarshalException ();
```

Remarks

The following table shows the initial property values for an instance of ContextMarshalException.

Expand table

Property	Value
InnerException	`null`.
Message	The empty string ("").

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ContextMarshalException(String)

Initializes a new instance of the ContextMarshalException class with a specified error message.

C#

```
public ContextMarshalException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

The following table shows the initial property values for an instance of ContextMarshalException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ContextMarshalException(SerializationInfo, StreamingContext)

Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ContextMarshalException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ContextMarshalException
```

```
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ContextMarshalException(String, Exception)

Initializes a new instance of the ContextMarshalException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public ContextMarshalException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ContextMarshalException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ContextStaticAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that the value of a static field is unique for a particular context.

C#

```
[System.AttributeUsage(System.AttributeTargets.Field, Inherited=false)]
public class ContextStaticAttribute : Attribute
```

Inheritance Object → Attribute → ContextStaticAttribute

Attributes AttributeUsageAttribute

Remarks

A static field marked with `ContextStaticAttribute` is not shared between contexts. If the indicated static field is accessed on a different context, it will contain a different value. Use this attribute as it is, and do not derive from it.

The following example shows the syntax of this attribute:

[ContextStatic]

static int f=7;

For more information about using attributes, see Attributes.

Constructors

Expand table

ContextStaticAttribute()	Initializes a new instance of the ContextStaticAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Attribute
- Context
- Extending Metadata Using Attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ContextStaticAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ContextStaticAttribute class.

C#

```
public ContextStaticAttribute ();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a base data type to another base data type.

C#

```
public static class Convert
```

Inheritance Object → Convert

Remarks

For more information about this API, see Supplemental API remarks for Convert.

Fields

Expand table

DBNull	A constant that represents a database column that is absent of data; that is, database null.

Methods

Expand table

ChangeType(Object, Type)	Returns an object of the specified type and whose value is equivalent to the specified object.
ChangeType(Object, Type, IFormatProvider)	Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.
ChangeType(Object, TypeCode)	Returns an object of the specified type whose value is

	equivalent to the specified object.
ChangeType(Object, TypeCode, IFormatProvider)	Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.
FromBase64CharArray(Char[], Int32, Int32)	Converts a subset of a Unicode character array, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array. Parameters specify the subset in the input array and the number of elements to convert.
FromBase64String(String)	Converts the specified string, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array.
FromHexString(ReadOnlySpan<Char>)	Converts the span, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.
FromHexString(String)	Converts the specified string, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.
GetTypeCode(Object)	Returns the TypeCode for the specified object.
IsDBNull(Object)	Returns an indication whether the specified object is of type DBNull.
ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32)	Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, and the number of elements in the input array to convert.
ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32, Base64FormattingOptions)	Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, the number of elements in the input array to convert, and whether line breaks are inserted in the output array.
ToBase64String(Byte[])	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits.
ToBase64String(Byte[], Base64FormattingOptions)	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. You can specify whether to insert line breaks in the return value.
ToBase64String(Byte[], Int32, Int32)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. Parameters specify the subset as an offset in the input array, and the number of elements in the array to convert.
ToBase64String(Byte[], Int32, Int32, Base64FormattingOptions)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64

	digits. Parameters specify the subset as an offset in the input array, the number of elements in the array to convert, and whether to insert line breaks in the return value.
ToBase64String(ReadOnlySpan<Byte>, Base64FormattingOptions)	Converts the 8-bit unsigned integers inside the specified read-only span into their equivalent string representation that is encoded with base-64 digits. You can optionally specify whether to insert line breaks in the return value.
ToBoolean(Boolean)	Returns the specified Boolean value; no actual conversion is performed.
ToBoolean(Byte)	Converts the value of the specified 8-bit unsigned integer to an equivalent Boolean value.
ToBoolean(Char)	Calling this method always throws InvalidCastException.
ToBoolean(DateTime)	Calling this method always throws InvalidCastException.
ToBoolean(Decimal)	Converts the value of the specified decimal number to an equivalent Boolean value.
ToBoolean(Double)	Converts the value of the specified double-precision floating-point number to an equivalent Boolean value.
ToBoolean(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent Boolean value.
ToBoolean(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent Boolean value.
ToBoolean(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent Boolean value.
ToBoolean(Object)	Converts the value of a specified object to an equivalent Boolean value.
ToBoolean(Object, IFormatProvider)	Converts the value of the specified object to an equivalent Boolean value, using the specified culture-specific formatting information.
ToBoolean(SByte)	Converts the value of the specified 8-bit signed integer to an equivalent Boolean value.
ToBoolean(Single)	Converts the value of the specified single-precision floating-point number to an equivalent Boolean value.
ToBoolean(String)	Converts the specified string representation of a logical value to its Boolean equivalent.
ToBoolean(String, IFormatProvider)	Converts the specified string representation of a logical value to its Boolean equivalent, using the specified culture-specific

	formatting information.
ToBoolean(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent Boolean value.
ToBoolean(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent Boolean value.
ToBoolean(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent Boolean value.
ToByte(Boolean)	Converts the specified Boolean value to the equivalent 8-bit unsigned integer.
ToByte(Byte)	Returns the specified 8-bit unsigned integer; no actual conversion is performed.
ToByte(Char)	Converts the value of the specified Unicode character to the equivalent 8-bit unsigned integer.
ToByte(DateTime)	Calling this method always throws InvalidCastException.
ToByte(Decimal)	Converts the value of the specified decimal number to an equivalent 8-bit unsigned integer.
ToByte(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 8-bit unsigned integer.
ToByte(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Object)	Converts the value of the specified object to an 8-bit unsigned integer.
ToByte(Object, IFormatProvider)	Converts the value of the specified object to an 8-bit unsigned integer, using the specified culture-specific formatting information.
ToByte(SByte)	Converts the value of the specified 8-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 8-bit unsigned integer.
ToByte(String)	Converts the specified string representation of a number to an equivalent 8-bit unsigned integer.

ToByte(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 8-bit unsigned integer, using specified culture-specific formatting information.
ToByte(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 8-bit unsigned integer.
ToByte(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToByte(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToByte(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToChar(Boolean)	Calling this method always throws InvalidCastException.
ToChar(Byte)	Converts the value of the specified 8-bit unsigned integer to its equivalent Unicode character.
ToChar(Char)	Returns the specified Unicode character value; no actual conversion is performed.
ToChar(DateTime)	Calling this method always throws InvalidCastException.
ToChar(Decimal)	Calling this method always throws InvalidCastException.
ToChar(Double)	Calling this method always throws InvalidCastException.
ToChar(Int16)	Converts the value of the specified 16-bit signed integer to its equivalent Unicode character.
ToChar(Int32)	Converts the value of the specified 32-bit signed integer to its equivalent Unicode character.
ToChar(Int64)	Converts the value of the specified 64-bit signed integer to its equivalent Unicode character.
ToChar(Object)	Converts the value of the specified object to a Unicode character.
ToChar(Object, IFormatProvider)	Converts the value of the specified object to its equivalent Unicode character, using the specified culture-specific formatting information.
ToChar(SByte)	Converts the value of the specified 8-bit signed integer to its equivalent Unicode character.
ToChar(Single)	Calling this method always throws InvalidCastException.

ToChar(String)	Converts the first character of a specified string to a Unicode character.
ToChar(String, IFormatProvider)	Converts the first character of a specified string to a Unicode character, using specified culture-specific formatting information.
ToChar(UInt16)	Converts the value of the specified 16-bit unsigned integer to its equivalent Unicode character.
ToChar(UInt32)	Converts the value of the specified 32-bit unsigned integer to its equivalent Unicode character.
ToChar(UInt64)	Converts the value of the specified 64-bit unsigned integer to its equivalent Unicode character.
ToDateTime(Boolean)	Calling this method always throws InvalidCastException.
ToDateTime(Byte)	Calling this method always throws InvalidCastException.
ToDateTime(Char)	Calling this method always throws InvalidCastException.
ToDateTime(DateTime)	Returns the specified DateTime object; no actual conversion is performed.
ToDateTime(Decimal)	Calling this method always throws InvalidCastException.
ToDateTime(Double)	Calling this method always throws InvalidCastException.
ToDateTime(Int16)	Calling this method always throws InvalidCastException.
ToDateTime(Int32)	Calling this method always throws InvalidCastException.
ToDateTime(Int64)	Calling this method always throws InvalidCastException.
ToDateTime(Object)	Converts the value of the specified object to a DateTime object.
ToDateTime(Object, IFormat Provider)	Converts the value of the specified object to a DateTime object, using the specified culture-specific formatting information.
ToDateTime(SByte)	Calling this method always throws InvalidCastException.
ToDateTime(Single)	Calling this method always throws InvalidCastException.
ToDateTime(String)	Converts the specified string representation of a date and time to an equivalent date and time value.
ToDateTime(String, IFormat Provider)	Converts the specified string representation of a number to an equivalent date and time, using the specified culture-specific formatting information.

ToDateTime(UInt16)	Calling this method always throws InvalidCastException.
ToDateTime(UInt32)	Calling this method always throws InvalidCastException.
ToDateTime(UInt64)	Calling this method always throws InvalidCastException.
ToDecimal(Boolean)	Converts the specified Boolean value to the equivalent decimal number.
ToDecimal(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent decimal number.
ToDecimal(Char)	Calling this method always throws InvalidCastException.
ToDecimal(DateTime)	Calling this method always throws InvalidCastException.
ToDecimal(Decimal)	Returns the specified decimal number; no actual conversion is performed.
ToDecimal(Double)	Converts the value of the specified double-precision floating-point number to an equivalent decimal number.
ToDecimal(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent decimal number.
ToDecimal(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent decimal number.
ToDecimal(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent decimal number.
ToDecimal(Object)	Converts the value of the specified object to an equivalent decimal number.
ToDecimal(Object, IFormat Provider)	Converts the value of the specified object to an equivalent decimal number, using the specified culture-specific formatting information.
ToDecimal(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent decimal number.
ToDecimal(Single)	Converts the value of the specified single-precision floating-point number to the equivalent decimal number.
ToDecimal(String)	Converts the specified string representation of a number to an equivalent decimal number.
ToDecimal(String, IFormat Provider)	Converts the specified string representation of a number to an equivalent decimal number, using the specified culture-specific formatting information.

ToDecimal(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent decimal number.
ToDecimal(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent decimal number.
ToDecimal(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent decimal number.
ToDouble(Boolean)	Converts the specified Boolean value to the equivalent double-precision floating-point number.
ToDouble(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent double-precision floating-point number.
ToDouble(Char)	Calling this method always throws InvalidCastException.
ToDouble(DateTime)	Calling this method always throws InvalidCastException.
ToDouble(Decimal)	Converts the value of the specified decimal number to an equivalent double-precision floating-point number.
ToDouble(Double)	Returns the specified double-precision floating-point number; no actual conversion is performed.
ToDouble(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent double-precision floating-point number.
ToDouble(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent double-precision floating-point number.
ToDouble(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent double-precision floating-point number.
ToDouble(Object)	Converts the value of the specified object to a double-precision floating-point number.
ToDouble(Object, IFormat Provider)	Converts the value of the specified object to an double-precision floating-point number, using the specified culture-specific formatting information.
ToDouble(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent double-precision floating-point number.
ToDouble(Single)	Converts the value of the specified single-precision floating-point number to an equivalent double-precision floating-point number.
ToDouble(String)	Converts the specified string representation of a number to an equivalent double-precision floating-point number.

ToDouble(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent double-precision floating-point number, using the specified culture-specific formatting information.
ToDouble(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent double-precision floating-point number.
ToDouble(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent double-precision floating-point number.
ToDouble(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent double-precision floating-point number.
ToHexString(Byte[])	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.
ToHexString(Byte[], Int32, Int32)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters. Parameters specify the subset as an offset in the input array and the number of elements in the array to convert.
ToHexString(ReadOnlySpan<Byte>)	Converts a span of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.
ToInt16(Boolean)	Converts the specified Boolean value to the equivalent 16-bit signed integer.
ToInt16(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit signed integer.
ToInt16(Char)	Converts the value of the specified Unicode character to the equivalent 16-bit signed integer.
ToInt16(DateTime)	Calling this method always throws InvalidCastException.
ToInt16(Decimal)	Converts the value of the specified decimal number to an equivalent 16-bit signed integer.
ToInt16(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 16-bit signed integer.
ToInt16(Int16)	Returns the specified 16-bit signed integer; no actual conversion is performed.
ToInt16(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 16-bit signed integer.
ToInt16(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 16-bit signed integer.

ToInt16(Object)	Converts the value of the specified object to a 16-bit signed integer.
ToInt16(Object, IFormatProvider)	Converts the value of the specified object to a 16-bit signed integer, using the specified culture-specific formatting information.
ToInt16(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 16-bit signed integer.
ToInt16(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 16-bit signed integer.
ToInt16(String)	Converts the specified string representation of a number to an equivalent 16-bit signed integer.
ToInt16(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 16-bit signed integer, using the specified culture-specific formatting information.
ToInt16(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 16-bit signed integer.
ToInt16(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 16-bit signed integer.
ToInt16(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit signed integer.
ToInt16(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit signed integer.
ToInt32(Boolean)	Converts the specified Boolean value to the equivalent 32-bit signed integer.
ToInt32(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit signed integer.
ToInt32(Char)	Converts the value of the specified Unicode character to the equivalent 32-bit signed integer.
ToInt32(DateTime)	Calling this method always throws InvalidCastException.
ToInt32(Decimal)	Converts the value of the specified decimal number to an equivalent 32-bit signed integer.
ToInt32(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 32-bit signed integer.
ToInt32(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 32-bit signed integer.

ToInt32(Int32)	Returns the specified 32-bit signed integer; no actual conversion is performed.
ToInt32(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 32-bit signed integer.
ToInt32(Object)	Converts the value of the specified object to a 32-bit signed integer.
ToInt32(Object, IFormatProvider)	Converts the value of the specified object to a 32-bit signed integer, using the specified culture-specific formatting information.
ToInt32(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 32-bit signed integer.
ToInt32(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 32-bit signed integer.
ToInt32(String)	Converts the specified string representation of a number to an equivalent 32-bit signed integer.
ToInt32(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 32-bit signed integer, using the specified culture-specific formatting information.
ToInt32(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 32-bit signed integer.
ToInt32(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit signed integer.
ToInt32(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 32-bit signed integer.
ToInt32(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit signed integer.
ToInt64(Boolean)	Converts the specified Boolean value to the equivalent 64-bit signed integer.
ToInt64(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit signed integer.
ToInt64(Char)	Converts the value of the specified Unicode character to the equivalent 64-bit signed integer.
ToInt64(DateTime)	Calling this method always throws InvalidCastException.
ToInt64(Decimal)	Converts the value of the specified decimal number to an equivalent 64-bit signed integer.

ToInt64(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 64-bit signed integer.
ToInt64(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 64-bit signed integer.
ToInt64(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 64-bit signed integer.
ToInt64(Int64)	Returns the specified 64-bit signed integer; no actual conversion is performed.
ToInt64(Object)	Converts the value of the specified object to a 64-bit signed integer.
ToInt64(Object, IFormatProvider)	Converts the value of the specified object to a 64-bit signed integer, using the specified culture-specific formatting information.
ToInt64(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 64-bit signed integer.
ToInt64(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 64-bit signed integer.
ToInt64(String)	Converts the specified string representation of a number to an equivalent 64-bit signed integer.
ToInt64(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 64-bit signed integer, using the specified culture-specific formatting information.
ToInt64(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 64-bit signed integer.
ToInt64(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit signed integer.
ToInt64(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit signed integer.
ToInt64(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 64-bit signed integer.
ToSByte(Boolean)	Converts the specified Boolean value to the equivalent 8-bit signed integer.
ToSByte(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 8-bit signed integer.
ToSByte(Char)	Converts the value of the specified Unicode character to the equivalent 8-bit signed integer.

ToSByte(DateTime)	Calling this method always throws InvalidCastException.
ToSByte(Decimal)	Converts the value of the specified decimal number to an equivalent 8-bit signed integer.
ToSByte(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 8-bit signed integer.
ToSByte(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 8-bit signed integer.
ToSByte(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 8-bit signed integer.
ToSByte(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 8-bit signed integer.
ToSByte(Object)	Converts the value of the specified object to an 8-bit signed integer.
ToSByte(Object, IFormatProvider)	Converts the value of the specified object to an 8-bit signed integer, using the specified culture-specific formatting information.
ToSByte(SByte)	Returns the specified 8-bit signed integer; no actual conversion is performed.
ToSByte(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 8-bit signed integer.
ToSByte(String)	Converts the specified string representation of a number to an equivalent 8-bit signed integer.
ToSByte(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 8-bit signed integer, using the specified culture-specific formatting information.
ToSByte(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 8-bit signed integer.
ToSByte(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 8-bit signed integer.
ToSByte(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit signed integer.
ToSByte(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit signed integer.
ToSingle(Boolean)	Converts the specified Boolean value to the equivalent single-precision floating-point number.

ToSingle(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent single-precision floating-point number.
ToSingle(Char)	Calling this method always throws InvalidCastException.
ToSingle(DateTime)	Calling this method always throws InvalidCastException.
ToSingle(Decimal)	Converts the value of the specified decimal number to an equivalent single-precision floating-point number.
ToSingle(Double)	Converts the value of the specified double-precision floating-point number to an equivalent single-precision floating-point number.
ToSingle(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent single-precision floating-point number.
ToSingle(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent single-precision floating-point number.
ToSingle(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent single-precision floating-point number.
ToSingle(Object)	Converts the value of the specified object to a single-precision floating-point number.
ToSingle(Object, IFormatProvider)	Converts the value of the specified object to an single-precision floating-point number, using the specified culture-specific formatting information.
ToSingle(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent single-precision floating-point number.
ToSingle(Single)	Returns the specified single-precision floating-point number; no actual conversion is performed.
ToSingle(String)	Converts the specified string representation of a number to an equivalent single-precision floating-point number.
ToSingle(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent single-precision floating-point number, using the specified culture-specific formatting information.
ToSingle(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent single-precision floating-point number.
ToSingle(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent single-precision floating-point number.
ToSingle(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent single-precision floating-point number.

ToString(Boolean)	Converts the specified Boolean value to its equivalent string representation.
ToString(Boolean, IFormat Provider)	Converts the specified Boolean value to its equivalent string representation.
ToString(Byte)	Converts the value of the specified 8-bit unsigned integer to its equivalent string representation.
ToString(Byte, IFormatProvider)	Converts the value of the specified 8-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Byte, Int32)	Converts the value of an 8-bit unsigned integer to its equivalent string representation in a specified base.
ToString(Char)	Converts the value of the specified Unicode character to its equivalent string representation.
ToString(Char, IFormatProvider)	Converts the value of the specified Unicode character to its equivalent string representation, using the specified culture-specific formatting information.
ToString(DateTime)	Converts the value of the specified DateTime to its equivalent string representation.
ToString(DateTime, IFormat Provider)	Converts the value of the specified DateTime to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Decimal)	Converts the value of the specified decimal number to its equivalent string representation.
ToString(Decimal, IFormat Provider)	Converts the value of the specified decimal number to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Double)	Converts the value of the specified double-precision floating-point number to its equivalent string representation.
ToString(Double, IFormat Provider)	Converts the value of the specified double-precision floating-point number to its equivalent string representation.
ToString(Int16)	Converts the value of the specified 16-bit signed integer to its equivalent string representation.
ToString(Int16, IFormatProvider)	Converts the value of the specified 16-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int16, Int32)	Converts the value of a 16-bit signed integer to its equivalent string representation in a specified base.

ToString(Int32)	Converts the value of the specified 32-bit signed integer to its equivalent string representation.
ToString(Int32, IFormatProvider)	Converts the value of the specified 32-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int32, Int32)	Converts the value of a 32-bit signed integer to its equivalent string representation in a specified base.
ToString(Int64)	Converts the value of the specified 64-bit signed integer to its equivalent string representation.
ToString(Int64, IFormatProvider)	Converts the value of the specified 64-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int64, Int32)	Converts the value of a 64-bit signed integer to its equivalent string representation in a specified base.
ToString(Object)	Converts the value of the specified object to its equivalent string representation.
ToString(Object, IFormatProvider)	Converts the value of the specified object to its equivalent string representation using the specified culture-specific formatting information.
ToString(SByte)	Converts the value of the specified 8-bit signed integer to its equivalent string representation.
ToString(SByte, IFormatProvider)	Converts the value of the specified 8-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Single)	Converts the value of the specified single-precision floating-point number to its equivalent string representation.
ToString(Single, IFormatProvider)	Converts the value of the specified single-precision floating-point number to its equivalent string representation, using the specified culture-specific formatting information.
ToString(String)	Returns the specified string instance; no actual conversion is performed.
ToString(String, IFormatProvider)	Returns the specified string instance; no actual conversion is performed.
ToString(UInt16)	Converts the value of the specified 16-bit unsigned integer to its equivalent string representation.
ToString(UInt16, IFormatProvider)	Converts the value of the specified 16-bit unsigned integer to its equivalent string representation, using the specified

	culture-specific formatting information.
ToString(UInt32)	Converts the value of the specified 32-bit unsigned integer to its equivalent string representation.
ToString(UInt32, IFormatProvider)	Converts the value of the specified 32-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(UInt64)	Converts the value of the specified 64-bit unsigned integer to its equivalent string representation.
ToString(UInt64, IFormatProvider)	Converts the value of the specified 64-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToUInt16(Boolean)	Converts the specified Boolean value to the equivalent 16-bit unsigned integer.
ToUInt16(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit unsigned integer.
ToUInt16(Char)	Converts the value of the specified Unicode character to the equivalent 16-bit unsigned integer.
ToUInt16(DateTime)	Calling this method always throws InvalidCastException.
ToUInt16(Decimal)	Converts the value of the specified decimal number to an equivalent 16-bit unsigned integer.
ToUInt16(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 16-bit unsigned integer.
ToUInt16(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 16-bit unsigned integer.
ToUInt16(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 16-bit unsigned integer.
ToUInt16(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 16-bit unsigned integer.
ToUInt16(Object)	Converts the value of the specified object to a 16-bit unsigned integer.
ToUInt16(Object, IFormat Provider)	Converts the value of the specified object to a 16-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt16(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 16-bit unsigned integer.

ToUInt16(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 16-bit unsigned integer.
ToUInt16(String)	Converts the specified string representation of a number to an equivalent 16-bit unsigned integer.
ToUInt16(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 16-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt16(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 16-bit unsigned integer.
ToUInt16(UInt16)	Returns the specified 16-bit unsigned integer; no actual conversion is performed.
ToUInt16(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit unsigned integer.
ToUInt16(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit unsigned integer.
ToUInt32(Boolean)	Converts the specified Boolean value to the equivalent 32-bit unsigned integer.
ToUInt32(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit unsigned integer.
ToUInt32(Char)	Converts the value of the specified Unicode character to the equivalent 32-bit unsigned integer.
ToUInt32(DateTime)	Calling this method always throws InvalidCastException.
ToUInt32(Decimal)	Converts the value of the specified decimal number to an equivalent 32-bit unsigned integer.
ToUInt32(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 32-bit unsigned integer.
ToUInt32(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 32-bit unsigned integer.
ToUInt32(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 32-bit unsigned integer.
ToUInt32(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 32-bit unsigned integer.
ToUInt32(Object)	Converts the value of the specified object to a 32-bit unsigned integer.

ToUInt32(Object, IFormat Provider)	Converts the value of the specified object to a 32-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt32(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 32-bit unsigned integer.
ToUInt32(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 32-bit unsigned integer.
ToUInt32(String)	Converts the specified string representation of a number to an equivalent 32-bit unsigned integer.
ToUInt32(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 32-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt32(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 32-bit unsigned integer.
ToUInt32(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit unsigned integer.
ToUInt32(UInt32)	Returns the specified 32-bit unsigned integer; no actual conversion is performed.
ToUInt32(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit unsigned integer.
ToUInt64(Boolean)	Converts the specified Boolean value to the equivalent 64-bit unsigned integer.
ToUInt64(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit unsigned integer.
ToUInt64(Char)	Converts the value of the specified Unicode character to the equivalent 64-bit unsigned integer.
ToUInt64(DateTime)	Calling this method always throws InvalidCastException.
ToUInt64(Decimal)	Converts the value of the specified decimal number to an equivalent 64-bit unsigned integer.
ToUInt64(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 64-bit unsigned integer.
ToUInt64(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 64-bit unsigned integer.
ToUInt64(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 64-bit unsigned integer.

ToUInt64(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 64-bit unsigned integer.
ToUInt64(Object)	Converts the value of the specified object to a 64-bit unsigned integer.
ToUInt64(Object, IFormat Provider)	Converts the value of the specified object to a 64-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt64(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 64-bit unsigned integer.
ToUInt64(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 64-bit unsigned integer.
ToUInt64(String)	Converts the specified string representation of a number to an equivalent 64-bit unsigned integer.
ToUInt64(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 64-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt64(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 64-bit unsigned integer.
ToUInt64(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit unsigned integer.
ToUInt64(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit unsigned integer.
ToUInt64(UInt64)	Returns the specified 64-bit unsigned integer; no actual conversion is performed.
TryFromBase64Chars(ReadOnly Span<Char>, Span<Byte>, Int32)	Tries to convert the specified span containing a string representation that is encoded with base-64 digits into a span of 8-bit unsigned integers.
TryFromBase64String(String, Span<Byte>, Int32)	Tries to convert the specified string representation that is encoded with base-64 digits into a span of 8-bit unsigned integers.
TryToBase64Chars(ReadOnly Span<Byte>, Span<Char>, Int32, Base64FormattingOptions)	Tries to convert the 8-bit unsigned integers inside the specified read-only span into their equivalent string representation that is encoded with base-64 digits. You can optionally specify whether to insert line breaks in the return value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IConvertible
- Object
- SByte
- Int16
- Int32
- Int64
- Byte
- UInt16
- UInt32
- UInt64
- Single
- Double
- Decimal
- Boolean
- Char
- String
- Type Conversion in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.DBNull Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A constant that represents a database column that is absent of data; that is, database null.

```
C#

public static readonly object DBNull;
```

Field Value

Object

Remarks

The DBNull field is equivalent to DBNull.Value, as the following example shows.

```
C#

Console.WriteLine(Convert.DBNull.Equals(DBNull.Value));
// Displays True.
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- DBNull
- Value

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ChangeType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an object of a specified type whose value is equivalent to a specified object.

Overloads

Expand table

ChangeType(Object, Type)	Returns an object of the specified type and whose value is equivalent to the specified object.
ChangeType(Object, TypeCode)	Returns an object of the specified type whose value is equivalent to the specified object.
ChangeType(Object, Type, IFormatProvider)	Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.
ChangeType(Object, TypeCode, IFormatProvider)	Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.

ChangeType(Object, Type)

Returns an object of the specified type and whose value is equivalent to the specified object.

C#

```
public static object? ChangeType (object? value, Type conversionType);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`conversionType` Type

The type of object to return.

Returns

Object

An object whose type is `conversionType` and whose value is equivalent to `value`.

-or-

A null reference (`Nothing` in Visual Basic), if `value` is `null` and `conversionType` is not a value type.

Exceptions

InvalidCastException

This conversion is not supported.

-or-

`value` is `null` and `conversionType` is a value type.

-or-

`value` does not implement the IConvertible interface.

FormatException

`value` is not in a format recognized by `conversionType`.

OverflowException

`value` represents a number that is out of the range of `conversionType`.

ArgumentNullException

`conversionType` is `null`.

Examples

The following example illustrates the use of the ChangeType method.

```
C#
```

```
using System;

public class ChangeTypeTest {
    public static void Main() {

        Double d = -2.345;
        int i = (int)Convert.ChangeType(d, typeof(int));

        Console.WriteLine("The double value {0} when converted to an int
becomes {1}", d, i);

        string s = "12/12/98";
        DateTime dt = (DateTime)Convert.ChangeType(s, typeof(DateTime));

        Console.WriteLine("The string value {0} when converted to a Date
becomes {1}", s, dt);
    }
}
```

Remarks

ChangeType is a general-purpose conversion method that converts the object specified by `value` to `conversionType`. The `value` parameter can be an object of any type, and `conversionType` can also be a Type object that represents any base or custom type. For the conversion to succeed, `value` must implement the IConvertible interface, because the method simply wraps a call to an appropriate IConvertible method. The method requires that conversion of `value` to `conversionType` be supported.

This method uses the current thread's culture for the conversion.

Notes to Callers

The ChangeType(Object, Type) method can convert an enumeration value to another type. However, it cannot convert another type to an enumeration value, even if the source type is the underlying type of the enumeration. To convert a type to an enumeration value, use a casting operator (in C#) or a conversion function (in Visual Basic). The following example illustrates the conversion to and from a `Continent` enumeration value.

C#

```
using System;

public enum Continent
```

```
{
   Africa, Antarctica, Asia, Australia, Europe,
   NorthAmerica, SouthAmerica
};

public class Example
{
   public static void Main()
   {
      // Convert a Continent to a Double.
      Continent cont = Continent.NorthAmerica;
      Console.WriteLine("{0:N2}",
                        Convert.ChangeType(cont, typeof(Double)));

      // Convert a Double to a Continent.
      Double number = 6.0;
      try {
         Console.WriteLine("{0}",
                           Convert.ChangeType(number,
typeof(Continent)));
      }
      catch (InvalidCastException) {
         Console.WriteLine("Cannot convert a Double to a Continent");
      }

      Console.WriteLine("{0}", (Continent) number);
   }
}
// The example displays the following output:
//       5.00
//       Cannot convert a Double to a Continent
//       SouthAmerica
```

The ChangeType(Object, Type) method can convert a nullable type to another type. However, it cannot convert another type to a value of a nullable type, even if `conversionType` is the underlying type of the Nullable<T>.To perform the conversion, you can use a casting operator (in C#) or a conversion function (in Visual Basic). The following example illustrates the conversion to and from a nullable type.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int? intValue1 = 12893;
      double dValue1 = (double) Convert.ChangeType(intValue1,
typeof(Double));
      Console.WriteLine("{0} ({1})--> {2} ({3})", intValue1,
intValue1.GetType().Name,
```

```
                        dValue1, dValue1.GetType().Name);

        float fValue1 = 16.3478f;
        int? intValue2 = (int) fValue1;
        Console.WriteLine("{0} ({1})--> {2} ({3})", fValue1,
fValue1.GetType().Name,
                        intValue2, intValue2.GetType().Name);
    }
}
// The example displays the following output:
//    12893 (Int32)--> 12893 (Double)
//    16.3478 (Single)--> 16 (Int32)
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ChangeType(Object, TypeCode)

Returns an object of the specified type whose value is equivalent to the specified object.

C#

```
public static object? ChangeType (object? value, TypeCode typeCode);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`typeCode` TypeCode

The type of object to return.

Returns

Object

An object whose underlying type is `typeCode` and whose value is equivalent to `value`.

-or-

A null reference (`Nothing` in Visual Basic), if `value` is `null` and `typeCode` is Empty, String, or Object.

Exceptions

InvalidCastException

This conversion is not supported.

-or-

`value` is `null` and `typeCode` specifies a value type.

-or-

`value` does not implement the IConvertible interface.

FormatException

`value` is not in a format recognized by the `typeCode` type.

OverflowException

`value` represents a number that is out of the range of the `typeCode` type.

ArgumentException

`typeCode` is invalid.

Examples

The following example illustrates how to use the ChangeType(Object, TypeCode) method to change an Object to the type specified by the TypeCode parameter, if possible.

```
C#

using System;

public class ChangeTypeTest {
```

```
    public static void Main() {

        Double d = -2.345;
        int i = (int)Convert.ChangeType(d, TypeCode.Int32);

        Console.WriteLine("The Double {0} when converted to an Int32 is
{1}", d, i);

        string s = "12/12/2009";
        DateTime dt = (DateTime)Convert.ChangeType(s, typeof(DateTime));

        Console.WriteLine("The String {0} when converted to a Date is
{1}", s, dt);
    }
}
// The example displays the following output:
//    The Double -2.345 when converted to an Int32 is -2
//    The String 12/12/2009 when converted to a Date is 12/12/2009
12:00:00 AM
```

Remarks

ChangeType(Object, TypeCode) is a general-purpose conversion method that converts the object specified by `value` to a predefined type specified by `typeCode`. The `value` parameter can be an object of any type. For the conversion to succeed, `value` must implement the IConvertible interface, because the method simply wraps a call to an appropriate IConvertible method. The method also requires that conversion of `value` to `typeCode` be supported.

The ChangeType(Object, TypeCode) method does not support the conversion of `value` to a custom type. To perform such a conversion, call the ChangeType(Object, Type) method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ChangeType(Object, Type, IFormatProvider)

Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.

```
C#

public static object? ChangeType (object? value, Type conversionType,
IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`conversionType` Type

The type of object to return.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Object

An object whose type is `conversionType` and whose value is equivalent to `value`.

-or-

`value`, if the Type of `value` and `conversionType` are equal.

-or-

A null reference (`Nothing` in Visual Basic), if `value` is `null` and `conversionType` is not a value type.

Exceptions

InvalidCastException

This conversion is not supported.

-or-

`value` is `null` and `conversionType` is a value type.

-or-

`value` does not implement the IConvertible interface.

FormatException

`value` is not in a format for `conversionType` recognized by `provider`.

OverflowException

`value` represents a number that is out of the range of `conversionType`.

ArgumentNullException

`conversionType` is `null`.

Examples

The following example defines a `Temperature` class that implements the IConvertible interface.

C#

```
using System;
using System.Globalization;

public class Temperature : IConvertible
{
   private decimal m_Temp;

   public Temperature(decimal temperature)
   {
      this.m_Temp = temperature;
   }

   public decimal Celsius
   {
      get { return this.m_Temp; }
   }

   public decimal Kelvin
   {
      get { return this.m_Temp + 273.15m; }
   }

   public decimal Fahrenheit
   {
      get { return Math.Round((decimal) (this.m_Temp * 9 / 5 + 32), 2); }
   }
```

```
    public override string ToString()
    {
        return m_Temp.ToString("N2") + "°C";
    }

    // IConvertible implementations.
    public TypeCode GetTypeCode()
    {
        return TypeCode.Object;
    }

    public bool ToBoolean(IFormatProvider provider)
    {
        if (m_Temp == 0)
            return false;
        else
            return true;
    }

    public byte ToByte(IFormatProvider provider)
    {
        if (m_Temp < Byte.MinValue || m_Temp > Byte.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Byte type.",
                                                      this.m_Temp));
        else
            return Decimal.ToByte(this.m_Temp);
    }

    public char ToChar(IFormatProvider provider)
    {
        throw new InvalidCastException("Temperature to Char conversion is
not supported.");
    }

    public DateTime ToDateTime(IFormatProvider provider)
    {
        throw new InvalidCastException("Temperature to DateTime conversion
is not supported.");
    }

    public decimal ToDecimal(IFormatProvider provider)
    {
        return this.m_Temp;
    }

    public double ToDouble(IFormatProvider provider)
    {
        return Decimal.ToDouble(this.m_Temp);
    }

    public short ToInt16(IFormatProvider provider)
    {
        if (this.m_Temp < Int16.MinValue || this.m_Temp > Int16.MaxValue)
```

```
            throw new OverflowException(String.Format("{0} is out of range
of the Int16 type.",
                                                        this.m_Temp));
         else
            return Decimal.ToInt16(this.m_Temp);
      }

      public int ToInt32(IFormatProvider provider)
         {
         if (this.m_Temp < Int32.MinValue || this.m_Temp > Int32.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Int32 type.",
                                                        this.m_Temp));
         else
            return Decimal.ToInt32(this.m_Temp);
      }

      public long ToInt64(IFormatProvider provider)
      {
         if (this.m_Temp < Int64.MinValue || this.m_Temp > Int64.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Int64 type.",
                                                        this.m_Temp));
         else
            return Decimal.ToInt64(this.m_Temp);
      }

      public sbyte ToSByte(IFormatProvider provider)
      {
         if (this.m_Temp < SByte.MinValue || this.m_Temp > SByte.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the SByte type.",
                                                        this.m_Temp));
         else
            return Decimal.ToSByte(this.m_Temp);
      }

      public float ToSingle(IFormatProvider provider)
      {
         return Decimal.ToSingle(this.m_Temp);
      }

      public string ToString(IFormatProvider provider)
      {
         return m_Temp.ToString("N2", provider) + "°C";
      }

      public object ToType(Type conversionType, IFormatProvider provider)
      {
         switch (Type.GetTypeCode(conversionType))
         {
            case TypeCode.Boolean:
               return this.ToBoolean(null);
            case TypeCode.Byte:
               return this.ToByte(null);
```

```
         case TypeCode.Char:
            return this.ToChar(null);
         case TypeCode.DateTime:
            return this.ToDateTime(null);
         case TypeCode.Decimal:
            return this.ToDecimal(null);
         case TypeCode.Double:
            return this.ToDouble(null);
         case TypeCode.Int16:
            return this.ToInt16(null);
         case TypeCode.Int32:
            return this.ToInt32(null);
         case TypeCode.Int64:
            return this.ToInt64(null);
         case TypeCode.Object:
            if (typeof(Temperature).Equals(conversionType))
               return this;
            else
               throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.",

conversionType.Name));
         case TypeCode.SByte:
            return this.ToSByte(null);
         case TypeCode.Single:
            return this.ToSingle(null);
         case TypeCode.String:
            return this.ToString(provider);
         case TypeCode.UInt16:
            return this.ToUInt16(null);
         case TypeCode.UInt32:
            return this.ToUInt32(null);
         case TypeCode.UInt64:
            return this.ToUInt64(null);
         default:
            throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
      }
   }

   public ushort ToUInt16(IFormatProvider provider)
   {
      if (this.m_Temp < UInt16.MinValue || this.m_Temp > UInt16.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt16 type.",
                                                     this.m_Temp));
      else
         return Decimal.ToUInt16(this.m_Temp);
   }

   public uint ToUInt32(IFormatProvider provider)
   {
      if (this.m_Temp < UInt32.MinValue || this.m_Temp > UInt32.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt32 type.",
```

```
                                                        this.m_Temp));
      else
         return Decimal.ToUInt32(this.m_Temp);
   }

   public ulong ToUInt64(IFormatProvider provider)
   {
      if (this.m_Temp < UInt64.MinValue || this.m_Temp > UInt64.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt64 type.",
                                                        this.m_Temp));
      else
         return Decimal.ToUInt64(this.m_Temp);
   }
}
```

The following example creates an instance of the `Temperature` class and calls the ChangeType(Object, Type, IFormatProvider) method to convert it to the basic numeric types supported by .NET and to a String. It illustrates that the ChangeType method wraps a call to the source type's IConvertible implementation.

C#

```
public class Example
{
   public static void Main()
   {
      Temperature cool = new Temperature(5);
      Type[] targetTypes = { typeof(SByte), typeof(Int16), typeof(Int32),
                             typeof(Int64), typeof(Byte), typeof(UInt16),
                             typeof(UInt32), typeof(UInt64),
typeof(Decimal),
                             typeof(Single), typeof(Double),
typeof(String) };
      CultureInfo provider = new CultureInfo("fr-FR");

      foreach (Type targetType in targetTypes)
      {
         try {
            object value = Convert.ChangeType(cool, targetType,
provider);
            Console.WriteLine("Converted {0} {1} to {2} {3}.",
                              cool.GetType().Name, cool.ToString(),
                              targetType.Name, value);
         }
         catch (InvalidCastException) {
            Console.WriteLine("Unsupported {0} --> {1} conversion.",
                              cool.GetType().Name, targetType.Name);
         }
         catch (OverflowException) {
            Console.WriteLine("{0} is out of range of the {1} type.",
                              cool, targetType.Name);
```

```
            }
        }
    }
}
// The example dosplays the following output:
//       Converted Temperature 5.00°C to SByte 5.
//       Converted Temperature 5.00°C to Int16 5.
//       Converted Temperature 5.00°C to Int32 5.
//       Converted Temperature 5.00°C to Int64 5.
//       Converted Temperature 5.00°C to Byte 5.
//       Converted Temperature 5.00°C to UInt16 5.
//       Converted Temperature 5.00°C to UInt32 5.
//       Converted Temperature 5.00°C to UInt64 5.
//       Converted Temperature 5.00°C to Decimal 5.
//       Converted Temperature 5.00°C to Single 5.
//       Converted Temperature 5.00°C to Double 5.
//       Converted Temperature 5.00°C to String 5,00°C.
```

Remarks

ChangeType is a general-purpose conversion method that converts the object specified by `value` to `conversionType`. The `value` parameter can be an object of any type, and `conversionType` can also be a Type object that represents any base or custom type. For the conversion to succeed, `value` must implement the IConvertible interface, because the method simply wraps a call to an appropriate IConvertible method. The method requires that conversion of `value` to `conversionType` be supported.

The `provider` parameter is an IFormatProvider implementation that supplies formatting information for the conversion. Whether and how this parameter is used depends on the underlying IConvertible implementation. If `value` is a base data type, `provider` is used only for the following conversions:

- Conversion from a number to a string, or from a string to a number. `provider` must be a CultureInfo object, a NumberFormatInfo object, or a custom IFormatProvider implementation that returns a NumberFormatInfo object. However, because the ChangeType(Object, TypeCode, IFormatProvider) method performs the conversion using the default "G" format specifier, the `provider` parameter has no effect if `value` or the target type is an unsigned integer. If `provider` is `null`, the CultureInfo object that represents the current culture is used.

- Conversion from a DateTime value to a string, or from a string to a DateTime value. `provider` must be a CultureInfo or DateTimeFormatInfo object. If

`provider` is `null`, the CultureInfo object that represents the current culture is used.

If `value` is an application-defined type, its IConvertible implementation may use the `provider` parameter.

Notes to Callers

The ChangeType(Object, Type, IFormatProvider) method can convert an enumeration value to another type. However, it cannot convert another type to an enumeration value, even if the source type is the underlying type of the enumeration. To convert a type to an enumeration value, use a casting operator (in C#) or a conversion function (in Visual Basic). The following example illustrates the conversion to and from a `Continent` enumeration value.

C#

```
using System;

public enum Continent
{
   Africa, Antarctica, Asia, Australia, Europe,
   NorthAmerica, SouthAmerica
};

public class Example
{
   public static void Main()
   {
      // Convert a Continent to a Double.
      Continent cont = Continent.NorthAmerica;
      Console.WriteLine("{0:N2}",
                        Convert.ChangeType(cont, typeof(Double)));

      // Convert a Double to a Continent.
      Double number = 6.0;
      try {
         Console.WriteLine("{0}",
                           Convert.ChangeType(number,
typeof(Continent)));
      }
      catch (InvalidCastException) {
         Console.WriteLine("Cannot convert a Double to a Continent");
      }

      Console.WriteLine("{0}", (Continent) number);
   }
}
// The example displays the following output:
```

```
//       5.00
//       Cannot convert a Double to a Continent
//       SouthAmerica
```

The ChangeType(Object, Type, IFormatProvider) method can convert a nullable type to another type. However, it cannot convert another type to a value of a nullable type, even if `conversionType` is the underlying type of the Nullable<T>. To perform the conversion, you can use a casting operator (in C#) or a conversion function (in Visual Basic). The following example illustrates the conversion to and from a nullable type.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int? intValue1 = 12893;
      double dValue1 = (double) Convert.ChangeType(intValue1,
typeof(Double), null);
      Console.WriteLine("{0} ({1})--> {2} ({3})", intValue1,
intValue1.GetType().Name,
                        dValue1, dValue1.GetType().Name);

      float fValue1 = 16.3478f;
      int? intValue2 = (int) fValue1;
      Console.WriteLine("{0} ({1})--> {2} ({3})", fValue1,
fValue1.GetType().Name,
                        intValue2, intValue2.GetType().Name);
   }
}
// The example displays the following output:
//    12893 (Int32)--> 12893 (Double)
//    16.3478 (Single)--> 16 (Int32)
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ChangeType(Object, TypeCode, IFormatProvider)

Returns an object of the specified type whose value is equivalent to the specified object. A parameter supplies culture-specific formatting information.

C#

```
public static object? ChangeType (object? value, TypeCode typeCode,
IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`typeCode` TypeCode

The type of object to return.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Object

An object whose underlying type is `typeCode` and whose value is equivalent to `value`.

-or-

A null reference (`Nothing` in Visual Basic), if `value` is `null` and `typeCode` is Empty, String, or Object.

Exceptions

InvalidCastException

This conversion is not supported.

-or-

`value` is `null` and `typeCode` specifies a value type.

-or-

`value` does not implement the IConvertible interface.

FormatException

`value` is not in a format for the `typeCode` type recognized by `provider`.

OverflowException

`value` represents a number that is out of the range of the `typeCode` type.

ArgumentException

`typeCode` is invalid.

Examples

The following example defines a custom format provider named `InterceptProvider` that indicates when its GetFormat method is called and returns a NumberFormatInfo for the fr-FR culture and a DateTimeFormatInfo object for the en-US culture. This format provider is used in all calls to the ChangeType(Object, TypeCode, IFormatProvider) method. The example then creates an array with a Double and a DateTime value and makes repeated calls to ChangeType(Object, TypeCode, IFormatProvider) with each value and each member of the TypeCode enumeration. The example illustrates when the method uses the IFormatProvider parameter and also illustrates the use of the `provider` parameter to perform culture-sensitive formatting.

C#

```
using System;
using System.Globalization;

public class InterceptProvider : IFormatProvider
{
   public object GetFormat(Type formatType)
   {
      if (formatType == typeof(NumberFormatInfo)) {
         Console.WriteLine("   Returning a fr-FR numeric format
```

```
provider.");
         return new System.Globalization.CultureInfo("fr-
FR").NumberFormat;
      }
      else if (formatType == typeof(DateTimeFormatInfo)) {
         Console.WriteLine("   Returning an en-US date/time format
provider.");
         return new System.Globalization.CultureInfo("en-
US").DateTimeFormat;
      }
      else {
         Console.WriteLine("   Requesting a format provider of {0}.",
formatType.Name);
         return null;
      }
   }
}

public class Example
{
   public static void Main()
   {
      object[] values = { 103.5d, new DateTime(2010, 12, 26, 14, 34, 0)
};
      IFormatProvider provider = new InterceptProvider();

      // Convert value to each of the types represented in TypeCode enum.
      foreach (object value in values)
      {
         // Iterate types in TypeCode enum.
         foreach (TypeCode enumType in ((TypeCode[])
Enum.GetValues(typeof(TypeCode))))
         {
            if (enumType == TypeCode.DBNull || enumType ==
TypeCode.Empty) continue;

            try {
               Console.WriteLine("{0} ({1}) --> {2} ({3}).",
                                 value, value.GetType().Name,
                                 Convert.ChangeType(value, enumType,
provider),
                                 enumType.ToString());
            }
            catch (InvalidCastException) {
               Console.WriteLine("Cannot convert a {0} to a {1}",
                                 value.GetType().Name,
enumType.ToString());
            }
            catch (OverflowException) {
               Console.WriteLine("Overflow: {0} is out of the range of a
{1}",
                                 value, enumType.ToString());
            }
         }
         Console.WriteLine();
```

```
        }
    }
}
// The example displays the following output:
//    103.5 (Double) --> 103.5 (Object).
//    103.5 (Double) --> True (Boolean).
//    Cannot convert a Double to a Char
//    103.5 (Double) --> 104 (SByte).
//    103.5 (Double) --> 104 (Byte).
//    103.5 (Double) --> 104 (Int16).
//    103.5 (Double) --> 104 (UInt16).
//    103.5 (Double) --> 104 (Int32).
//    103.5 (Double) --> 104 (UInt32).
//    103.5 (Double) --> 104 (Int64).
//    103.5 (Double) --> 104 (UInt64).
//    103.5 (Double) --> 103.5 (Single).
//    103.5 (Double) --> 103.5 (Double).
//    103.5 (Double) --> 103.5 (Decimal).
//    Cannot convert a Double to a DateTime
//       Returning a fr-FR numeric format provider.
//    103.5 (Double) --> 103,5 (String).
//
//    12/26/2010 2:34:00 PM (DateTime) --> 12/26/2010 2:34:00 PM
(Object).
//    Cannot convert a DateTime to a Boolean
//    Cannot convert a DateTime to a Char
//    Cannot convert a DateTime to a SByte
//    Cannot convert a DateTime to a Byte
//    Cannot convert a DateTime to a Int16
//    Cannot convert a DateTime to a UInt16
//    Cannot convert a DateTime to a Int32
//    Cannot convert a DateTime to a UInt32
//    Cannot convert a DateTime to a Int64
//    Cannot convert a DateTime to a UInt64
//    Cannot convert a DateTime to a Single
//    Cannot convert a DateTime to a Double
//    Cannot convert a DateTime to a Decimal
//    12/26/2010 2:34:00 PM (DateTime) --> 12/26/2010 2:34:00 PM
(DateTime).
//       Returning an en-US date/time format provider.
//    12/26/2010 2:34:00 PM (DateTime) --> 12/26/2010 2:34:00 PM
(String).
```

Remarks

ChangeType(Object, TypeCode, IFormatProvider) is a general-purpose conversion method that converts the object specified by `value` to a predefined type specified by `typeCode`. The `value` parameter can be an object of any type. For the conversion to succeed, `value` must implement the IConvertible interface, because the method

simply wraps a call to an appropriate IConvertible method. The method also requires that conversion of `value` to `typeCode` be supported.

The ChangeType(Object, TypeCode, IFormatProvider) method does not support the conversion of `value` to a custom type. To perform such a conversion, call the ChangeType(Object, Type, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation that supplies formatting information for the conversion. Whether and how this parameter is used depends on the underlying IConvertible implementation. If `value` is a base data type, `provider` is used only for the following conversions. If a `null` IFormatProvider argument is passed to these methods, the CultureInfo object that represents the current culture is used.

- Conversion from a number to a string, or from a string to a number. `provider` must be a CultureInfo object, a NumberFormatInfo object, or a custom IFormatProvider implementation that returns a NumberFormatInfo object. However, because the ChangeType(Object, TypeCode, IFormatProvider) method performs the conversion using the default "G" format specifier, the `provider` parameter has no effect if `value` or the target type is an unsigned integer.
- Conversion from a DateTime value to a string, or from a string to a DateTime value. `provider` must be a CultureInfo or DateTimeFormatInfo object.

If `value` is an application-defined type, its IConvertible implementation may use the `provider` parameter.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Convert.FromBase64CharArray(Char[], Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a subset of a Unicode character array, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array. Parameters specify the subset in the input array and the number of elements to convert.

```
C#

public static byte[] FromBase64CharArray (char[] inArray, int offset, int
length);
```

Parameters

`inArray` Char[]

A Unicode character array.

`offset` Int32

A position within `inArray`.

`length` Int32

The number of elements in `inArray` to convert.

Returns

Byte[]

An array of 8-bit unsigned integers equivalent to `length` elements at position `offset` in `inArray`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentOutOfRangeException

`offset` or `length` is less than 0.

-or-

`offset` plus `length` indicates a position not within `inArray`.

FormatException

The length of `inArray`, ignoring white-space characters, is not zero or a multiple of 4.

-or-

The format of `inArray` is invalid. `inArray` contains a non-base-64 character, more than two padding characters, or a non-white-space character among the padding characters.

Examples

The following example demonstrates the use of the FromBase64CharArray(Char[], Int32, Int32) method to decode UUencoded (base-64) data and save it as binary output.

C#

```
public void DecodeWithCharArray() {
   System.IO.StreamReader inFile;
   char[] base64CharArray;

   try {
      inFile = new System.IO.StreamReader(inputFileName,
                              System.Text.Encoding.ASCII);
      base64CharArray = new char[inFile.BaseStream.Length];
      inFile.Read(base64CharArray, 0, (int)inFile.BaseStream.Length);
      inFile.Close();
   }
   catch (System.Exception exp) {
      // Error creating stream or reading from it.
      System.Console.WriteLine("{0}", exp.Message);
      return;
   }

   // Convert the Base64 UUEncoded input into binary output.
   byte[] binaryData;
   try {
      binaryData =
         System.Convert.FromBase64CharArray(base64CharArray,
                                            0,
                                            base64CharArray.Length);
```

```
    }
    catch ( System.ArgumentNullException ) {
       System.Console.WriteLine("Base 64 character array is null.");
       return;
    }
    catch ( System.FormatException ) {
       System.Console.WriteLine("Base 64 Char Array length is not " +
          "4 or is not an even multiple of 4." );
       return;
    }

    // Write out the decoded data.
    System.IO.FileStream outFile;
    try {
       outFile = new System.IO.FileStream(outputFileName,
                                          System.IO.FileMode.Create,
                                          System.IO.FileAccess.Write);
       outFile.Write(binaryData, 0, binaryData.Length);
       outFile.Close();
    }
    catch (System.Exception exp) {
       // Error creating stream or writing to it.
       System.Console.WriteLine("{0}", exp.Message);
    }
 }
```

The following example demonstrates the ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32, Base64FormattingOptions) and FromBase64CharArray(Char[], Int32, Int32) methods. The input is divided into groups of three bytes (24 bits) each. Consequently, each group consists of four 6-bit numbers where each number ranges from decimal 0 to 63. In this example, there are 85 3-byte groups with one byte remaining. The first group consists of the hexadecimal values 00, 01, and 02, which yield four 6-bit values equal to decimal 0, 0, 4, and 2. Those four values correspond to the base-64 digits, "A", "A", "E", and "C", at the beginning of the output.

If an integral number of 3-byte groups does not exist, the remaining bytes are effectively padded with zeros to form a complete group. In this example, the value of the last byte is hexadecimal FF. The first 6 bits are equal to decimal 63, which corresponds to the base-64 digit "/" at the end of the output, and the next 2 bits are padded with zeros to yield decimal 48, which corresponds to the base-64 digit, "w". The last two 6-bit values are padding and correspond to the valueless padding character, "=".

C#

```
// This example demonstrates the Convert.ToBase64CharArray() and
//                                Convert.FromBase64CharArray methods
```

```
using System;

class Sample
{
    public static void Main()
    {
    byte[] byteArray1 = new byte[256];
    byte[] byteArray2 = new byte[256];
    char[] charArray  = new char[352];
    int charArrayLength;
    string nl = Environment.NewLine;

    string ruler1a = "         1         2         3         4";
    string ruler2a = "1234567890123456789012345678901234567890";
    string ruler3a = "----+----+----+----+----+----+----+----+";
    string ruler1b = "         5         6         7      ";
    string ruler2b = "123456789012345678901234567890123456";
    string ruler3b = "----+----+----+----+----+----+----+-";
    string ruler   = String.Concat(ruler1a, ruler1b, nl,
                                   ruler2a, ruler2b, nl,
                                   ruler3a, ruler3b);

// 1) Initialize and display a Byte array of arbitrary data.
    Console.WriteLine("1) Input: A Byte array of arbitrary data.{0}", nl);
    for (int x = 0; x < byteArray1.Length; x++)
    {
    byteArray1[x] = (byte)x;
    Console.Write("{0:X2} ", byteArray1[x]);
    if (((x+1)%20) == 0) Console.WriteLine();
    }
    Console.Write("{0}{0}", nl);

// 2) Convert the input Byte array to a Char array, with newlines inserted.
    charArrayLength =
        Convert.ToBase64CharArray(byteArray1, 0, byteArray1.Length,
                                   charArray, 0,
Base64FormattingOptions.InsertLineBreaks);
    Console.WriteLine("2) Convert the input Byte array to a Char array with
newlines.");
    Console.Write("   Output: A Char array (length = {0}). ",
charArrayLength);
    Console.WriteLine("The elements of the array are:{0}", nl);
    Console.WriteLine(ruler);
    Console.WriteLine(new String(charArray));
    Console.WriteLine();

// 3) Convert the Char array back to a Byte array.
    Console.WriteLine("3) Convert the Char array to an output Byte array.");
    byteArray2 = Convert.FromBase64CharArray(charArray, 0, charArrayLength);

// 4) Are the input and output Byte arrays equivalent?
    Console.WriteLine("4) The output Byte array is equal to the input Byte
array: {0}",
                      ArraysAreEqual(byteArray1, byteArray2));
    }
```

```
    public static bool ArraysAreEqual(byte[] a1, byte[] a2)
    {
    if (a1.Length != a2.Length) return false;
    for (int i = 0; i < a1.Length; i++)
        if (a1[i] != a2[i]) return false;
    return true;
    }
}
/*
This example produces the following results:

1) Input: A Byte array of arbitrary data.

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B
8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3
B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7
C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB
DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

2) Convert the input Byte array to a Char array with newlines.
   Output: A Char array (length = 352). The elements of the array are:

         1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012345678
----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+-
AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vMDEyMzQ1Njc4
OTo7PD0+P0BBQkNERUZHSElKS0xNTk9QUVJTVFVWV1hZWltcXV5fYGFiY2RlZmdoaWprbG1ub3Bx
cnN0dXZ3eHl6e3x9fn+AgYKDhIWGh4iJiouMjY6PkJGSk5SVlpeYmZqbnJ2en6ChoqOkpaanqKmq
q6ytrq+wsbKztLW2t7i5uru8vb6/wMHCw8TFxsfIycrLzM3Oz9DR0tPU1dbX2Nna29zd3t/g4eLj
5OXm5+jp6uvs7e7v8PHy8/T19vf4+fr7/P3+/w==

3) Convert the Char array to an output Byte array.
4) The output Byte array is equal to the input Byte array: True

*/
```

Remarks

`inArray` is composed of base-64 digits, white-space characters, and trailing padding characters. The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", lowercase characters "a" to "z", numerals "0" to "9", and the symbols "+" and "/".

The white-space characters, and their Unicode names and hexadecimal code points, are tab (CHARACTER TABULATION, U+0009), newline (LINE FEED, U+000A), carriage return (CARRIAGE RETURN, U+000D), and blank (SPACE, U+0020). An arbitrary number of white-space characters can appear in `inArray` because all white-space characters are ignored.

The valueless character, "=", is used for trailing padding. The end of `inArray` can consist of zero, one, or two padding characters.

> **Important**
>
> The **FromBase64CharArray** method is designed to process a single character array that contains all the data to be decoded. To decode base-64 character data from a stream, use the **System.Security.Cryptography.FromBase64Transform** class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32)
- FromBase64Transform

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Convert.FromBase64String(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array.

C#

```
public static byte[] FromBase64String (string s);
```

Parameters

s String

The string to convert.

Returns

Byte[]

An array of 8-bit unsigned integers that is equivalent to `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

The length of `s`, ignoring white-space characters, is not zero or a multiple of 4.

-or-

The format of `s` is invalid. `s` contains a non-base-64 character, more than two padding characters, or a non-white space-character among the padding characters.

Examples

The following example uses the ToBase64String(Byte[]) method to convert a byte array to a UUencoded (base-64) string, and then calls the FromBase64String(String) method to restore the original byte array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
       // Define a byte array.
       byte[] bytes = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
       Console.WriteLine("The byte array: ");
       Console.WriteLine("   {0}\n", BitConverter.ToString(bytes));

       // Convert the array to a base 64 string.
       string s = Convert.ToBase64String(bytes);
       Console.WriteLine("The base 64 string:\n   {0}\n", s);

       // Restore the byte array.
       byte[] newBytes = Convert.FromBase64String(s);
       Console.WriteLine("The restored byte array: ");
       Console.WriteLine("   {0}\n", BitConverter.ToString(newBytes));
   }
}
// The example displays the following output:
//     The byte array:
//        02-04-06-08-0A-0C-0E-10-12-14
//
//     The base 64 string:
//        AgQGCAoMDhASFA==
//
//     The restored byte array:
//        02-04-06-08-0A-0C-0E-10-12-14
```

The following is a more complex example that creates a 20-element array of 32-bit integers. It then uses the BitConverter.GetBytes(Int32) method to convert each element into a byte array, which it stores in the appropriate position in a buffer by calling the Array.Copy(Array, Int32, Array, Int32, Int32) method. This buffer is then passed to the ToBase64String(Byte[]) method to create a UUencoded (base-64) string. It then calls the FromBase64String(String) method to decode the UUencoded string, and calls the BitConverter.ToInt32 method to convert each set of four bytes (the size of a 32-bit integer) to an integer. The output from the example shows that the original array has been successfully restored.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define an array of 20 elements and display it.
      int[] arr = new int[20];
      int value = 1;
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++) {
         arr[ctr] = value;
         value = value * 2 + 1;
      }
      DisplayArray(arr);

      // Convert the array of integers to a byte array.
      byte[] bytes = new byte[arr.Length * 4];
      for (int ctr = 0; ctr < arr.Length; ctr++) {
         Array.Copy(BitConverter.GetBytes(arr[ctr]), 0,
                    bytes, ctr * 4, 4);
      }

      // Encode the byte array using Base64 encoding
      String base64 = Convert.ToBase64String(bytes);
      Console.WriteLine("The encoded string: ");
      for (int ctr = 0; ctr <= base64.Length / 50; ctr++)
         Console.WriteLine(base64.Substring(ctr * 50,
                                            ctr * 50 + 50 <= base64.Length
                                               ? 50 : base64.Length - ctr *
50));
      Console.WriteLine();

      // Convert the string back to a byte array.
      byte[] newBytes = Convert.FromBase64String(base64);

      // Convert the byte array back to an integer array.
      int[] newArr = new int[newBytes.Length/4];
      for (int ctr = 0; ctr < newBytes.Length / 4; ctr ++)
         newArr[ctr] = BitConverter.ToInt32(newBytes, ctr * 4);

      DisplayArray(newArr);
   }

   private static void DisplayArray(Array arr)
   {
      Console.WriteLine("The array:");
      Console.Write("{ ");
      for (int ctr = 0; ctr < arr.GetUpperBound(0); ctr++) {
         Console.Write("{0}, ", arr.GetValue(ctr));
         if ((ctr + 1) % 10 == 0)
            Console.Write("\n  ");
      }
```

```
        Console.WriteLine("{0} {1}", arr.GetValue(arr.GetUpperBound(0)), "}");
        Console.WriteLine();
    }
}
// The example displays the following output:
// The array:
// { 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023,
//   2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287, 1048575
}
//
// The encoded string:
// AQAAAAMAAAAHAAAADwAAAB8AAAA/AAAAfwAAAP8AAAD/AQAA/w
// MAAP8HAAD/DwAA/x8AAP8/AAD/fwAA//8AAP//AQD//wMA//8H
//
// The array:
// { 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023,
//   2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287, 1048575
}
```

Remarks

`s` is composed of base-64 digits, white-space characters, and trailing padding characters. The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", lowercase characters "a" to "z", numerals "0" to "9", and the symbols "+" and "/".

The white-space characters, and their Unicode names and hexadecimal code points, are tab (CHARACTER TABULATION, U+0009), newline (LINE FEED, U+000A), carriage return (CARRIAGE RETURN, U+000D), and blank (SPACE, U+0020). An arbitrary number of white-space characters can appear in `s` because all white-space characters are ignored.

The valueless character, "=", is used for trailing padding. The end of `s` can consist of zero, one, or two padding characters.

> **ⓘ Important**
>
> The **FromBase64String** method is designed to process a single string that contains all the data to be decoded. To decode base-64 character data from a stream, use the **System.Security.Cryptography.FromBase64Transform** class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToBase64String(Byte[])
- FromBase64Transform

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.FromHexString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

FromHexString(ReadOnlySpan<Char>)	Converts the span, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.
FromHexString(String)	Converts the specified string, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.

FromHexString(ReadOnlySpan<Char>)

Converts the span, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.

C#

```
public static byte[] FromHexString (ReadOnlySpan<char> chars);
```

Parameters

`chars` ReadOnlySpan<Char>

The span to convert.

Returns

Byte[]

An array of 8-bit unsigned integers that is equivalent to `chars`.

Exceptions

FormatException

The length of `chars`, is not zero or a multiple of 2.

FormatException

The format of `chars` is invalid. `chars` contains a non-hex character.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

FromHexString(String)

Converts the specified string, which encodes binary data as hex characters, to an equivalent 8-bit unsigned integer array.

```
C#

public static byte[] FromHexString (string s);
```

Parameters

s String

The string to convert.

Returns

Byte[]

An array of 8-bit unsigned integers that is equivalent to `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

The length of `s`, is not zero or a multiple of 2.

FormatException

The format of `s` is invalid. `s` contains a non-hex character.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.GetTypeCode(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for the specified object.

C#

```
public static TypeCode GetTypeCode (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

Returns

TypeCode

The TypeCode for `value`, or Empty if `value` is `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Convert.IsDBNull(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an indication whether the specified object is of type DBNull.

C#

```
public static bool IsDBNull (object? value);
```

Parameters

value Object

An object.

Returns

Boolean

`true` if `value` is of type DBNull; otherwise, `false`.

Examples

The following example uses a SqlDataReader object to retrieve survey data from a database. It assigns each row's field values to an array, and then passes each array element to the IsDBNull method. If the method returns `true`, the example assigns the string "NA" to the array element. The array is then added to the Rows collection of a System.Windows.Forms.DataGridView control.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
   // Define ADO.NET objects.
   SqlConnection conn = new SqlConnection(connectionString);
   SqlCommand cmd = new SqlCommand();
   SqlDataReader dr;
```

```
   // Open connection, and retrieve dataset.
   conn.Open();

   // Define Command object.
   cmd.CommandText = "Select * From Responses";
   cmd.CommandType = CommandType.Text;
   cmd.Connection = conn;

   // Retrieve data reader.
   dr = cmd.ExecuteReader();

   int fieldCount = dr.FieldCount;
   object[] fieldValues = new object[fieldCount];
   string[] headers = new string[fieldCount];

   // Get names of fields.
   for (int ctr = 0; ctr < fieldCount; ctr++)
      headers[ctr] = dr.GetName(ctr);

   // Set up data grid.
   this.grid.ColumnCount = fieldCount;

   this.grid.ColumnHeadersDefaultCellStyle.BackColor = Color.Navy;
   this.grid.ColumnHeadersDefaultCellStyle.ForeColor = Color.White;
   this.grid.ColumnHeadersDefaultCellStyle.Font = new Font(this.grid.Font,
FontStyle.Bold);

   this.grid.AutoSizeRowsMode =
DataGridViewAutoSizeRowsMode.DisplayedCellsExceptHeaders;
   this.grid.ColumnHeadersBorderStyle =
DataGridViewHeaderBorderStyle.Single;
   this.grid.CellBorderStyle = DataGridViewCellBorderStyle.Single;
   this.grid.GridColor = Color.Black;
   this.grid.RowHeadersVisible = true;

   for (int columnNumber = 0; columnNumber < headers.Length;
columnNumber++)
      this.grid.Columns[columnNumber].Name = headers[columnNumber];

   // Get data, replace missing values with "NA", and display it.
   while (dr.Read())
   {
      dr.GetValues(fieldValues);

      for (int fieldCounter = 0; fieldCounter < fieldCount; fieldCounter++)
      {
         if (Convert.IsDBNull(fieldValues[fieldCounter]))
            fieldValues[fieldCounter] = "NA";
      }
      grid.Rows.Add(fieldValues);
   }
   dr.Close();
}
```

Remarks

The IsDBNull method tests whether the `value` parameter is equal to DBNull.Value. It is equivalent to the following code:

```
C#
return DBNull.Value.Equals(value);
```

> **Note**
>
> **DBNull.Value** is used to indicate a value that is missing. It is not equivalent to `null` or to **String.Empty**. Therefore, code such as `Convert.IsDBNull(null)` in C# or `Convert.IsDBNull(Nothing)` in Visual Basic returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- DBNull

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToBase64CharArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits.

Overloads

Expand table

ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32, Base64FormattingOptions)	Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, the number of elements in the input array to convert, and whether line breaks are inserted in the output array.
ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32)	Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, and the number of elements in the input array to convert.

ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32, Base64FormattingOptions)

Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, the number of elements in the input array to convert, and whether line breaks are inserted in the output array.

C#

```
public static int ToBase64CharArray (byte[] inArray, int offsetIn, int
length, char[] outArray, int offsetOut, Base64FormattingOptions options);
```

Parameters

`inArray` Byte[]

An input array of 8-bit unsigned integers.

`offsetIn` Int32

A position within `inArray`.

`length` Int32

The number of elements of `inArray` to convert.

`outArray` Char[]

An output array of Unicode characters.

`offsetOut` Int32

A position within `outArray`.

`options` Base64FormattingOptions

InsertLineBreaks to insert a line break every 76 characters, or None to not insert line breaks.

Returns

Int32

A 32-bit signed integer containing the number of bytes in `outArray`.

Exceptions

ArgumentNullException

`inArray` or `outArray` is `null`.

ArgumentOutOfRangeException

`offsetIn`, `offsetOut`, or `length` is negative.

-or-

`offsetIn` plus `length` is greater than the length of `inArray`.

-or-

`offsetOut` plus the number of elements to return is greater than the length of `outArray`.

ArgumentException

`options` is not a valid Base64FormattingOptions value.

Examples

The following example demonstrates the ToBase64CharArray method. The input is divided into groups of three bytes (24 bits) each. Consequently, each group consists of four 6-bit numbers where each number ranges from decimal 0 to 63. In this example, there are 85 3-byte groups with one byte remaining. The first group consists of the hexadecimal values 00, 01, and 02, which yield four 6-bit values equal to decimal 0, 0, 4, and 2. Those four values correspond to the base-64 digits "A", "A", "E", and "C" at the beginning of the output.

If an integral number of 3-byte groups does not exist, the remaining bytes are effectively padded with zeros to form a complete group. In this example, the value of the last byte is hexadecimal FF. The first 6 bits are equal to decimal 63, which corresponds to the base-64 digit "/" at the end of the output, and the next 2 bits are padded with zeros to yield decimal 48, which corresponds to the base-64 digit, "w". The last two 6-bit values are padding and correspond to the valueless padding character, "=".

C#

```
// This example demonstrates the Convert.ToBase64CharArray() and
//                                Convert.FromBase64CharArray methods

using System;

class Sample
{
    public static void Main()
    {
    byte[] byteArray1 = new byte[256];
    byte[] byteArray2 = new byte[256];
    char[] charArray  = new char[352];
    int charArrayLength;
    string nl = Environment.NewLine;

    string ruler1a = "         1         2         3         4";
    string ruler2a = "1234567890123456789012345678901234567890";
    string ruler3a = "----+----+----+----+----+----+----+----+";
    string ruler1b = "         5         6         7      ";
    string ruler2b = "123456789012345678901234567890123456";
```

```
    string ruler3b = "----+----+----+----+----+----+----+-";
    string ruler   = String.Concat(ruler1a, ruler1b, nl,
                                   ruler2a, ruler2b, nl,
                                   ruler3a, ruler3b);

// 1) Initialize and display a Byte array of arbitrary data.
    Console.WriteLine("1) Input: A Byte array of arbitrary data.{0}",
nl);
    for (int x = 0; x < byteArray1.Length; x++)
    {
    byteArray1[x] = (byte)x;
    Console.Write("{0:X2} ", byteArray1[x]);
    if (((x+1)%20) == 0) Console.WriteLine();
    }
    Console.Write("{0}{0}", nl);

// 2) Convert the input Byte array to a Char array, with newlines
inserted.
    charArrayLength =
        Convert.ToBase64CharArray(byteArray1, 0, byteArray1.Length,
                                   charArray, 0,
Base64FormattingOptions.InsertLineBreaks);
    Console.WriteLine("2) Convert the input Byte array to a Char array
with newlines.");
    Console.Write("   Output: A Char array (length = {0}). ",
charArrayLength);
    Console.WriteLine("The elements of the array are:{0}", nl);
    Console.WriteLine(ruler);
    Console.WriteLine(new String(charArray));
    Console.WriteLine();

// 3) Convert the Char array back to a Byte array.
    Console.WriteLine("3) Convert the Char array to an output Byte
array.");
    byteArray2 = Convert.FromBase64CharArray(charArray, 0,
charArrayLength);

// 4) Are the input and output Byte arrays equivalent?
    Console.WriteLine("4) The output Byte array is equal to the input
Byte array: {0}",
                      ArraysAreEqual(byteArray1, byteArray2));
    }

    public static bool ArraysAreEqual(byte[] a1, byte[] a2)
    {
    if (a1.Length != a2.Length) return false;
    for (int i = 0; i < a1.Length; i++)
        if (a1[i] != a2[i]) return false;
    return true;
    }
}
/*
This example produces the following results:

1) Input: A Byte array of arbitrary data.
```

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B
8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3
B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7
C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB
DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

2) Convert the input Byte array to a Char array with newlines.
   Output: A Char array (length = 352). The elements of the array are:

         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123
456
----+----+----+----+----+----+----+----+----+----+----+----+----+----+---
-+-
AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vMDEyMzQ1N
jc4
OTo7PD0+P0BBQkNERUZHSElKS0xNTk9QUVJTVFVWV1hZWltcXV5fYGFiY2RlZmdoaWprbG1ub
3Bx
cnN0dXZ3eHl6e3x9fn+AgYKDhIWGh4iJiouMjY6PkJGSk5SVlpeYmZqbnJ2en6ChoqOkpaanq
Kmq
q6ytrq+wsbKztLW2t7i5uru8vb6/wMHCw8TFxsfIycrLzM3Oz9DR0tPU1dbX2Nna29zd3t/g4
eLj
5OXm5+jp6uvs7e7v8PHy8/T19vf4+fr7/P3+/w==

3) Convert the Char array to an output Byte array.
4) The output Byte array is equal to the input Byte array: True

*/
```

Remarks

The subset of `length` elements of the `inArray` parameter starting at position `offsetIn`, are taken as a numeric value and converted to a subset of elements in the `outArray` parameter starting at position `offsetOut`. The return value indicates the number of converted elements in `outArray`. The subset of `outArray` consists of base-64 digits.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character "=" is used for trailing padding.

The `offset` and `length` parameters are 32-bit signed numbers. The `offsetIn` and `offsetOut` parameters are zero-based array positions.

> **Important**
>
> The **ToBase64CharArray** method is designed to process a single byte array that contains all the data to be encoded. To create a base-64 character array from a byte stream, use the **System.Security.Cryptography.ToBase64Transform** class.

If the `options` parameter is set to InsertLineBreaks and the output of the conversion is longer than 76 characters, a line break is inserted every 76 characters. A line break is defined as a carriage return character (U+000D) followed by a line feed character (U+000A). For more information, see RFC 2045, "Multipurpose Internet Mail Extensions", at https://www.rfc-editor.org/.

See also

- FromBase64CharArray(Char[], Int32, Int32)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToBase64CharArray(Byte[], Int32, Int32, Char[], Int32)

Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, and the number of elements in the input array to convert.

```C#
public static int ToBase64CharArray (byte[] inArray, int offsetIn, int length, char[] outArray, int offsetOut);
```

Parameters

`inArray` Byte[]

An input array of 8-bit unsigned integers.

`offsetIn` Int32

A position within `inArray`.

`length` Int32

The number of elements of `inArray` to convert.

`outArray` Char[]

An output array of Unicode characters.

`offsetOut` Int32

A position within `outArray`.

Returns

Int32

A 32-bit signed integer containing the number of bytes in `outArray`.

Exceptions

ArgumentNullException

`inArray` or `outArray` is `null`.

ArgumentOutOfRangeException

`offsetIn`, `offsetOut`, or `length` is negative.

-or-

`offsetIn` plus `length` is greater than the length of `inArray`.

-or-

`offsetOut` plus the number of elements to return is greater than the length of `outArray`.

Examples

The following example demonstrates using the ToBase64CharArray method to UUencode (encode in base 64) a binary stream, then save the encoding to a file.

C#

```
public void EncodeWithCharArray() {
   System.IO.FileStream inFile;
   byte[]             binaryData;

   try {
      inFile = new System.IO.FileStream(inputFileName,
                                        System.IO.FileMode.Open,
                                       System.IO.FileAccess.Read);
      binaryData = new Byte[inFile.Length];
      long bytesRead = inFile.Read(binaryData, 0,
                                   (int) inFile.Length);
      inFile.Close();
   }
   catch (System.Exception exp) {
      // Error creating stream or reading from it.
      System.Console.WriteLine("{0}", exp.Message);
      return;
   }

   // Convert the binary input into Base64 UUEncoded output.
   // Each 3 byte sequence in the source data becomes a 4 byte
   // sequence in the character array.
   long arrayLength = (long) ((4.0d/3.0d) * binaryData.Length);

   // If array length is not divisible by 4, go up to the next
   // multiple of 4.
   if (arrayLength % 4 != 0) {
      arrayLength += 4 - arrayLength % 4;
   }

   char[] base64CharArray = new char[arrayLength];
   try {
      System.Convert.ToBase64CharArray(binaryData,
                                       0,
                                       binaryData.Length,
                                       base64CharArray,
                                       0);
   }
   catch (System.ArgumentNullException) {
      System.Console.WriteLine("Binary data array is null.");
      return;
   }
```

```
    catch (System.ArgumentOutOfRangeException) {
       System.Console.WriteLine("Char Array is not large enough.");
       return;
    }

    // Write the UUEncoded version to the output file.
    System.IO.StreamWriter outFile;
    try {
       outFile = new System.IO.StreamWriter(outputFileName,
                                false,
                                System.Text.Encoding.ASCII);
       outFile.Write(base64CharArray);
       outFile.Close();
    }
    catch (System.Exception exp) {
       // Error creating stream or writing to it.
       System.Console.WriteLine("{0}", exp.Message);
    }
 }
```

Remarks

The subset of `length` elements of `inArray` starting at position `offsetIn`, are taken as a numeric value and converted to a subset of elements in `outArray` starting at position `offsetOut`. The return value indicates the number of converted elements in `outArray`. The subset of `outArray` consists of base-64 digits.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character, "=", is used for trailing padding.

The `offset` and `length` parameters are 32-bit signed numbers. The `offsetIn` and `offsetOut` parameters are zero-based array positions.

> ⓘ **Important**
>
> The **ToBase64CharArray** method is designed to process a single byte array that contains all the data to be encoded. To create a base-64 character array from a byte stream, use the **System.Security.Cryptography.ToBase64Transform** class.

See also

- FromBase64CharArray(Char[], Int32, Int32)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToBase64String Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits.

Overloads

Expand table

ToBase64String(Byte[])	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits.
ToBase64String(Byte[], Base64FormattingOptions)	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. You can specify whether to insert line breaks in the return value.
ToBase64String(ReadOnlySpan<Byte>, Base64FormattingOptions)	Converts the 8-bit unsigned integers inside the specified read-only span into their equivalent string representation that is encoded with base-64 digits. You can optionally specify whether to insert line breaks in the return value.
ToBase64String(Byte[], Int32, Int32)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. Parameters specify the subset as an offset in the input array, and the number of elements in the array to convert.
ToBase64String(Byte[], Int32, Int32, Base64FormattingOptions)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. Parameters specify the subset as an offset in the input array, the number of elements in the array to convert, and whether to insert line breaks in the return value.

ToBase64String(Byte[])

Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits.

```
C#

public static string ToBase64String (byte[] inArray);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

Returns

String

The string representation, in base 64, of the contents of `inArray`.

Exceptions

ArgumentNullException

`inArray` is `null`.

Examples

The following example uses the ToBase64String(Byte[]) method to convert a byte array to a UUencoded (base-64) string, and then calls the FromBase64String(String) method to restore the original byte array.

```
C#

using System;

public class Example
{
   public static void Main()
   {
       // Define a byte array.
       byte[] bytes = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
       Console.WriteLine("The byte array: ");
       Console.WriteLine("   {0}\n", BitConverter.ToString(bytes));

       // Convert the array to a base 64 string.
```

```
        string s = Convert.ToBase64String(bytes);
        Console.WriteLine("The base 64 string:\n   {0}\n", s);

        // Restore the byte array.
        byte[] newBytes = Convert.FromBase64String(s);
        Console.WriteLine("The restored byte array: ");
        Console.WriteLine("   {0}\n", BitConverter.ToString(newBytes));
    }
}
// The example displays the following output:
//     The byte array:
//        02-04-06-08-0A-0C-0E-10-12-14
//
//     The base 64 string:
//        AgQGCAoMDhASFA==
//
//     The restored byte array:
//        02-04-06-08-0A-0C-0E-10-12-14
```

The following is a more complex example that creates a 20-element array of 32-bit integers. It then uses the BitConverter.GetBytes(Int32) method to convert each element into a byte array, which it stores in the appropriate position in a buffer by calling the Array.Copy(Array, Int32, Array, Int32, Int32) method. This buffer is then passed to the ToBase64String(Byte[]) method to create a UUencoded (base-64) string. It then calls the FromBase64String(String) method to decode the UUencoded string, and calls the BitConverter.ToInt32 method to convert each set of four bytes (the size of a 32-bit integer) to an integer. The output from the example shows that the original array has been successfully restored.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define an array of 20 elements and display it.
      int[] arr = new int[20];
      int value = 1;
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++) {
         arr[ctr] = value;
         value = value * 2 + 1;
      }
      DisplayArray(arr);

      // Convert the array of integers to a byte array.
      byte[] bytes = new byte[arr.Length * 4];
      for (int ctr = 0; ctr < arr.Length; ctr++) {
         Array.Copy(BitConverter.GetBytes(arr[ctr]), 0,
```

```
                    bytes, ctr * 4, 4);
      }

      // Encode the byte array using Base64 encoding
      String base64 = Convert.ToBase64String(bytes);
      Console.WriteLine("The encoded string: ");
      for (int ctr = 0; ctr <= base64.Length / 50; ctr++)
         Console.WriteLine(base64.Substring(ctr * 50,
                                            ctr * 50 + 50 <=
base64.Length
                                               ? 50 : base64.Length - ctr
* 50));
      Console.WriteLine();

      // Convert the string back to a byte array.
      byte[] newBytes = Convert.FromBase64String(base64);

      // Convert the byte array back to an integer array.
      int[] newArr = new int[newBytes.Length/4];
      for (int ctr = 0; ctr < newBytes.Length / 4; ctr ++)
         newArr[ctr] = BitConverter.ToInt32(newBytes, ctr * 4);

      DisplayArray(newArr);
   }

   private static void DisplayArray(Array arr)
   {
      Console.WriteLine("The array:");
      Console.Write("{ ");
      for (int ctr = 0; ctr < arr.GetUpperBound(0); ctr++) {
         Console.Write("{0}, ", arr.GetValue(ctr));
         if ((ctr + 1) % 10 == 0)
            Console.Write("\n  ");
      }
      Console.WriteLine("{0} {1}", arr.GetValue(arr.GetUpperBound(0)),
"}");
      Console.WriteLine();
   }
}
// The example displays the following output:
// The array:
// { 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023,
//   2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287,
1048575 }
//
// The encoded string:
// AQAAAAMAAAAHAAAADwAAAB8AAAA/AAAAfwAAAP8AAAD/AQAA/w
// MAAP8HAAD/DwAA/x8AAP8/AAD/fwAA//8AAP//AQD//wMA//8H
//
// The array:
// { 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023,
//   2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287,
1048575 }
```

Remarks

The elements of `inArray` are taken as a numeric value and converted to a string representation that is encoded with base-64 digits.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character, "=", is used for trailing padding.

> ⓘ **Important**
>
> The **ToBase64String** method is designed to process a single byte array that contains all the data to be encoded. To encode data from a stream, use the **System.Security.Cryptography.ToBase64Transform** class.

Ordinarily, the ToBase64String method is not used to round-trip a UUEncoded (base-64 encoded) string. That is, if you decode a string by calling the FromBase64String method, then encode the returned byte array by calling the ToBase64String method, the resulting string will not necessarily be identical to the original string. The string will round-trip only if the original string is a valid base-64 encoded string.

See also

- FromBase64String(String)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBase64String(Byte[], Base64FormattingOptions)

Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. You can specify whether to insert line breaks in the return value.

```
C#

public static string ToBase64String (byte[] inArray,
Base64FormattingOptions options);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

`options` Base64FormattingOptions

InsertLineBreaks to insert a line break every 76 characters, or None to not insert line breaks.

Returns

String

The string representation in base 64 of the elements in `inArray`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentException

`options` is not a valid Base64FormattingOptions value.

Examples

The following example calls the Convert.ToBase64String(Byte[], Base64FormattingOptions) with a Base64FormattingOptions.InsertLineBreaks argument to insert line breaks in the string that is produced by encoding a 100-element byte array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
       // Define a byte array.
       var bytes = new byte[100];
       int originalTotal = 0;
       for (int ctr = 0; ctr <= bytes.GetUpperBound(0); ctr++) {
          bytes[ctr] = (byte)(ctr + 1);
          originalTotal += bytes[ctr];
       }
       // Display summary information about the array.
       Console.WriteLine("The original byte array:");
       Console.WriteLine("   Total elements: {0}", bytes.Length);
       Console.WriteLine("   Length of String Representation: {0}",
                         BitConverter.ToString(bytes).Length);
       Console.WriteLine("   Sum of elements: {0:N0}", originalTotal);
       Console.WriteLine();

       // Convert the array to a base 64 string.
       string s = Convert.ToBase64String(bytes,

Base64FormattingOptions.InsertLineBreaks);
       Console.WriteLine("The base 64 string:\n   {0}\n", s);

       // Restore the byte array.
       Byte[] newBytes = Convert.FromBase64String(s);
       int newTotal = 0;
       foreach (var newByte in newBytes)
          newTotal += newByte;

       // Display summary information about the restored array.
       Console.WriteLine("   Total elements: {0}", newBytes.Length);
       Console.WriteLine("   Length of String Representation: {0}",
                         BitConverter.ToString(newBytes).Length);
       Console.WriteLine("   Sum of elements: {0:N0}", newTotal);
   }
}
// The example displays the following output:
//   The original byte array:
//      Total elements: 100
//      Length of String Representation: 299
//      Sum of elements: 5,050
//
//   The base 64 string:
//
AQIDBAUGBwgJCgsMDQ4PEBESExQVFhcYGRobHB0eHyAhIiMkJSYnKCkqKywtLi8wMTIzNDU2N
zg5
//   Ojs8PT4/QEFCQ0RFRkdISUpLTE1OT1BRUlNUVVZXWFlaW1xdXl9gYWJjZA==
//
```

```
//      Total elements: 100
//      Length of String Representation: 299
//      Sum of elements: 5,050
```

As the output from the example shows, the Convert.FromBase64String succeeds in restoring the original byte array; the line break characters are ignored during the conversion.

Remarks

The elements of the `inArray` parameter are taken as a numeric value and converted to a string representation in base 64.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character "=" is used for trailing padding.

> **Important**
>
> The **ToBase64String** method is designed to process a single byte array that contains all the data to be encoded. To encode data from a stream, use the **System.Security.Cryptography.ToBase64Transform** class.

If the `options` parameter is set to InsertLineBreaks and the output of the conversion is longer than 76 characters, a line break is inserted every 76 characters. A line break is defined as a carriage return character (U+000D) followed by a line feed character (U+000A). Because line breaks are considered white-space characters in a base-64 encoding, they are ignored when converting a base-64 encoded string back to a byte array. The line breaks are simply convenient when displaying the encoded string to a control or a device such as a console window. For more information, see RFC 2045, "Multipurpose Internet Mail Extensions", at https://www.rfc-editor.org/.

See also

- FromBase64String(String)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToBase64String(ReadOnlySpan<Byte>, Base64FormattingOptions)

Converts the 8-bit unsigned integers inside the specified read-only span into their equivalent string representation that is encoded with base-64 digits. You can optionally specify whether to insert line breaks in the return value.

C#

```
public static string ToBase64String (ReadOnlySpan<byte> bytes,
Base64FormattingOptions options = System.Base64FormattingOptions.None);
```

Parameters

`bytes` ReadOnlySpan<Byte>

A read-only span of 8-bit unsigned integers.

`options` Base64FormattingOptions

One of the enumeration values that specify whether to insert line breaks in the return value. The default value is None.

Returns

String

The string representation in base 64 of the elements in `inArray`. If the length of `bytes` is 0, an empty string is returned.

Exceptions

ArgumentException

`options` is not a valid Base64FormattingOptions value.

OutOfMemoryException

The output length was larger than Int32.MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ToBase64String(Byte[], Int32, Int32)

Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. Parameters specify the subset as an offset in the input array, and the number of elements in the array to convert.

```
C#

public static string ToBase64String (byte[] inArray, int offset, int
length);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

`offset` Int32

An offset in `inArray`.

`length` Int32

The number of elements of `inArray` to convert.

Returns

String

The string representation in base 64 of `length` elements of `inArray`, starting at position `offset`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentOutOfRangeException

`offset` or `length` is negative.

-or-

`offset` plus `length` is greater than the length of `inArray`.

Remarks

The elements of `inArray` are taken as a numeric value and converted to a string representation in base 64.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character, "=", is used for trailing padding.

The `offset` and `length` parameters are 32-bit signed numbers. The `offset` parameter is zero-based.

> ⓘ **Important**
>
> The **ToBase64String** method is designed to process a single byte array that contains all the data to be encoded. To encode data from a stream, use the **System.Security.Cryptography.ToBase64Transform** class.

See also

- FromBase64String(String)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBase64String(Byte[], Int32, Int32, Base64FormattingOptions)

Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. Parameters specify the subset as an offset in the input array, the number of elements in the array to convert, and whether to insert line breaks in the return value.

C#

```
public static string ToBase64String (byte[] inArray, int offset, int
length, Base64FormattingOptions options);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

`offset` Int32

An offset in `inArray`.

`length` Int32

The number of elements of `inArray` to convert.

`options` Base64FormattingOptions

InsertLineBreaks to insert a line break every 76 characters, or None to not insert line breaks.

Returns

String

The string representation in base 64 of `length` elements of `inArray`, starting at position `offset`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentOutOfRangeException

`offset` or `length` is negative.

-or-

`offset` plus `length` is greater than the length of `inArray`.

ArgumentException

`options` is not a valid Base64FormattingOptions value.

Examples

The following example demonstrates the ToBase64String method. The input is divided into groups of three bytes (24 bits) each. Consequently, each group consists of four 6-bit numbers where each number ranges from decimal 0 to 63. In this example, there are 85 3-byte groups with one byte remaining. The first group consists of the hexadecimal values 00, 01, and 02, which yield four 6-bit values equal to decimal 0, 0, 4, and 2. Those four values correspond to the base-64 digits "A", "A", "E", and "C" at the beginning of the output.

If an integral number of 3-byte groups does not exist, the remaining bytes are effectively padded with zeros to form a complete group. In this example, the value of the last byte is hexadecimal FF. The first 6 bits are equal to decimal 63, which corresponds to the base-64 digit "/" at the end of the output, and the next 2 bits are padded with zeros to yield decimal 48, which corresponds to the base-64 digit, "w". The last two 6-bit values are padding and correspond to the valueless padding character, "=".

C#

```
// This example demonstrates the Convert.ToBase64String() and
//                                Convert.FromBase64String() methods
```

```
using System;
class Sample
{
    public static void Main()
    {
    byte[] inArray  = new byte[256];
    byte[] outArray = new byte[256];
    string s2;
    string s3;
    string step1 = "1) The input is a byte array (inArray) of arbitrary
data.";
    string step2 = "2) Convert a subarray of the input data array to a
base 64 string.";
    string step3 = "3) Convert the entire input data array to a base 64
string.";
    string step4 = "4) The two methods in steps 2 and 3 produce the same
result: {0}";
    string step5 = "5) Convert the base 64 string to an output byte array
(outArray).";
    string step6 = "6) The input and output arrays, inArray and outArray,
are equal: {0}";
    int x;
    string nl = Environment.NewLine;
    string ruler1a = "         1         2         3         4";
    string ruler2a = "1234567890123456789012345678901234567890";
    string ruler3a = "----+----+----+----+----+----+----+----+";
    string ruler1b = "         5         6         7      ";
    string ruler2b = "123456789012345678901234567890123456";
    string ruler3b = "----+----+----+----+----+----+----+-";
    string ruler   = String.Concat(ruler1a, ruler1b, nl,
                                   ruler2a, ruler2b, nl,
                                   ruler3a, ruler3b, nl);

// 1) Display an arbitrary array of input data (inArray). The data could
be
//    derived from user input, a file, an algorithm, etc.

    Console.WriteLine(step1);
    Console.WriteLine();
    for (x = 0; x < inArray.Length; x++)
        {
        inArray[x] = (byte)x;
        Console.Write("{0:X2} ", inArray[x]);
        if (((x+1)%20) == 0) Console.WriteLine();
        }
    Console.Write("{0}{0}", nl);

// 2) Convert a subarray of the input data to a base64 string. In this
case,
//    the subarray is the entire input data array. New lines (CRLF) are
inserted.

    Console.WriteLine(step2);
    s2 = Convert.ToBase64String(inArray, 0, inArray.Length,
```

```
Base64FormattingOptions.InsertLineBreaks);
    Console.WriteLine("{0}{1}{2}{3}", nl, ruler, s2, nl);

// 3) Convert the input data to a base64 string. In this case, the entire
//    input data array is converted by default. New lines (CRLF) are
inserted.

    Console.WriteLine(step3);
    s3 = Convert.ToBase64String(inArray,
Base64FormattingOptions.InsertLineBreaks);

// 4) Test whether the methods in steps 2 and 3 produce the same result.
    Console.WriteLine(step4, s2.Equals(s3));

// 5) Convert the base 64 string to an output array (outArray).
    Console.WriteLine(step5);
    outArray = Convert.FromBase64String(s2);

// 6) Is outArray equal to inArray?
   Console.WriteLine(step6, ArraysAreEqual(inArray, outArray));
   }

    public static bool ArraysAreEqual(byte[] a1, byte[] a2)
    {
    if (a1.Length != a2.Length) return false;
    for (int i = 0; i < a1.Length; i++)
        if (a1[i] != a2[i]) return false;
    return true;
    }
}
/*
This example produces the following results:

1) The input is a byte array (inArray) of arbitrary data.

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B
8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3
B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7
C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB
DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

2) Convert a subarray of the input data array to a base 64 string.

         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123
456
----+----+----+----+----+----+----+----+----+----+----+----+----+----+---
```

```
-+-
AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vMDEyMzQ1N
jc4
OTo7PD0+P0BBQkNERUZHSElKS0xNTk9QUVJTVFVWV1hZWltcXV5fYGFiY2RlZmdoaWprbG1ub
3Bx
cnN0dXZ3eHl6e3x9fn+AgYKDhIWGh4iJiouMjY6PkJGSk5SVlpeYmZqbnJ2en6ChoqOkpaanq
Kmq
q6ytrq+wsbKztLW2t7i5uru8vb6/wMHCw8TFxsfIycrLzM3Oz9DR0tPU1dbX2Nna29zd3t/g4
eLj
5OXm5+jp6uvs7e7v8PHy8/T19vf4+fr7/P3+/w==

3) Convert the entire input data array to a base 64 string.
4) The two methods in steps 2 and 3 produce the same result: True
5) Convert the base 64 string to an output byte array (outArray).
6) The input and output arrays, inArray and outArray, are equal: True

*/
```

Remarks

The elements of the `inArray` parameter are taken as a numeric value and converted to a string representation in base 64.

The base-64 digits in ascending order from zero are the uppercase characters "A" to "Z", the lowercase characters "a" to "z", the numerals "0" to "9", and the symbols "+" and "/". The valueless character "=" is used for trailing padding.

The `offset` and `length` parameters are 32-bit signed numbers. The `offset` parameter is zero-based.

> **Important**
>
> The **ToBase64String** method is designed to process a single byte array that contains all the data to be encoded. To encode data from a stream, use the **System.Security.Cryptography.ToBase64Transform** class.

If the `options` parameter is set to InsertLineBreaks and the output of the conversion is longer than 76 characters, a line break is inserted every 76 characters. A line break is defined as a carriage return character (U+000D) followed by a line feed character (U+000A). For more information, see RFC 2045, "Multipurpose Internet Mail Extensions", at https://www.rfc-editor.org/.

See also

- FromBase64String(String)
- ToBase64Transform

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToBoolean Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to an equivalent Boolean value.

Overloads

Expand table

ToBoolean(SByte)	Converts the value of the specified 8-bit signed integer to an equivalent Boolean value.
ToBoolean(String, IFormatProvider)	Converts the specified string representation of a logical value to its Boolean equivalent, using the specified culture-specific formatting information.
ToBoolean(Object, IFormatProvider)	Converts the value of the specified object to an equivalent Boolean value, using the specified culture-specific formatting information.
ToBoolean(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent Boolean value.
ToBoolean(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent Boolean value.
ToBoolean(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent Boolean value.
ToBoolean(String)	Converts the specified string representation of a logical value to its Boolean equivalent.
ToBoolean(Single)	Converts the value of the specified single-precision floating-point number to an equivalent Boolean value.
ToBoolean(Object)	Converts the value of a specified object to an equivalent Boolean value.
ToBoolean(DateTime)	Calling this method always throws InvalidCastException.
ToBoolean(Int32)	Converts the value of the specified 32-bit signed integer to an

	equivalent Boolean value.
ToBoolean(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent Boolean value.
ToBoolean(Double)	Converts the value of the specified double-precision floating-point number to an equivalent Boolean value.
ToBoolean(Decimal)	Converts the value of the specified decimal number to an equivalent Boolean value.
ToBoolean(Char)	Calling this method always throws InvalidCastException.
ToBoolean(Byte)	Converts the value of the specified 8-bit unsigned integer to an equivalent Boolean value.
ToBoolean(Boolean)	Returns the specified Boolean value; no actual conversion is performed.
ToBoolean(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent Boolean value.

ToBoolean(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to an equivalent Boolean value.

C#

```
[System.CLSCompliant(false)]
public static bool ToBoolean (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of SByte values to Boolean values.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, 100, SByte.MaxValue };
bool result;

foreach (sbyte number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-5}  -->  {1}", number, result);
}
// The example displays the following output:
//       -128   -->  True
//       -1     -->  True
//       0      -->  False
//       10     -->  True
//       100    -->  True
//       127    -->  True
```

See also

- SByte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(String, IFormatProvider)

Converts the specified string representation of a logical value to its Boolean equivalent, using the specified culture-specific formatting information.

```
C#

public static bool ToBoolean (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the value of either TrueString or FalseString.

`provider` IFormatProvider

An object that supplies culture-specific formatting information. This parameter is ignored.

Returns

Boolean

`true` if `value` equals TrueString, or `false` if `value` equals FalseString or `null`.

Exceptions

FormatException

`value` is not equal to TrueString or FalseString.

Remarks

If you prefer not to handle an exception if the conversion fails, you can call the Boolean.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Object, IFormatProvider)

Converts the value of the specified object to an equivalent Boolean value, using the specified culture-specific formatting information.

C#

```
public static bool ToBoolean (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Boolean

`true` or `false`, which reflects the value returned by invoking the ToBoolean(IFormatProvider) method for the underlying type of `value`. If `value` is `null`, the method returns `false`.

Exceptions

FormatException

`value` is a string that does not equal TrueString or FalseString.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion of `value` to a Boolean is not supported.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the ToBoolean(Object, IFormatProvider) method. This method returns `true` if any of the non-discarded array values are non-zero. The IFormatProvider object determines how elements are discarded for this calculation.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }

    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
```

```
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;

    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
        {
            if( index >= 0 && index < data.Count )
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
```

```
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;

        if( data.Count == 0 )
            return 0.0;

        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];

                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];

            default:
                return 0.0;
        }
    }

    // Get the AverageInfo object from the caller's format provider,
```

```
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }

    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
            GetAverageInfo( provider ).TypeOfAverage )
        {
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
```

```
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
            Byte.MinValue, Byte.MaxValue, provider ) );
    }

    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
    }

    public double ToDouble( IFormatProvider provider )
    {
```

```
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }
```

```
    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}

class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );

         // Display the dataset elements.
        if( ds.Count > 0 )
        {
            Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
            for( int iX = 1; iX < ds.Count; iX++ )
                Console.Write( ", {0}", ds[ iX ] );
            Console.WriteLine( "]\n" );
        }

        Console.WriteLine( fmt, "Convert.", "Default",
            "Geometric Mean", "Median");
        Console.WriteLine( fmt, "--------", "-------",
            "--------------", "------");
        Console.WriteLine( fmt, "ToBoolean",
            Convert.ToBoolean( ds, null ),
            Convert.ToBoolean( ds, geMean ),
            Convert.ToBoolean( ds, median ) );
        Console.WriteLine( fmt, "ToByte",
            Convert.ToByte( ds, null ),
            Convert.ToByte( ds, geMean ),
            Convert.ToByte( ds, median ) );
        Console.WriteLine( fmt, "ToChar",
            Convert.ToChar( ds, null ),
            Convert.ToChar( ds, geMean ),
            Convert.ToChar( ds, median ) );
        Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
            "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
            "ToDateTime", Convert.ToDateTime( ds, null ),
```

```
            Convert.ToDateTime( ds, geMean ),
            Convert.ToDateTime( ds, median ) );
        Console.WriteLine( fmt, "ToDecimal",
            Convert.ToDecimal( ds, null ),
            Convert.ToDecimal( ds, geMean ),
            Convert.ToDecimal( ds, median ) );
        Console.WriteLine( fmt, "ToDouble",
            Convert.ToDouble( ds, null ),
            Convert.ToDouble( ds, geMean ),
            Convert.ToDouble( ds, median ) );
        Console.WriteLine( fmt, "ToInt16",
            Convert.ToInt16( ds, null ),
            Convert.ToInt16( ds, geMean ),
            Convert.ToInt16( ds, median ) );
        Console.WriteLine( fmt, "ToInt32",
            Convert.ToInt32( ds, null ),
            Convert.ToInt32( ds, geMean ),
            Convert.ToInt32( ds, median ) );
        Console.WriteLine( fmt, "ToInt64",
            Convert.ToInt64( ds, null ),
            Convert.ToInt64( ds, geMean ),
            Convert.ToInt64( ds, median ) );
        Console.WriteLine( fmt, "ToSByte",
            Convert.ToSByte( ds, null ),
            Convert.ToSByte( ds, geMean ),
            Convert.ToSByte( ds, median ) );
        Console.WriteLine( fmt, "ToSingle",
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );

        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );
```

```
        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       103                  59                  67
ToChar                         g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                      103                  59                  67
ToInt32                      103                  59                  67
ToInt64                      103                  59                  67
ToSByte                      103                  59                  67
ToSingle                 103.4917            59.43321                67.3
ToUInt16                     103                  59                  67
ToUInt32                     103                  59                  67
ToUInt64                     103                  59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal              750999.89    631577.237188435            499999.5
ToDouble               750999.89    631577.237188435            499999.5
ToInt16                    32767               32767               32767
ToInt32                   751000              631577              500000
ToInt64                   751000              631577              500000
ToSByte                      127                 127                 127
ToSingle                750999.9            631577.3            499999.5
ToUInt16                   65535               65535               65535
ToUInt32                  751000              631577              500000
ToUInt64                  751000              631577              500000
*/
```

Remarks

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. The base types ignore the `provider` parameter; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

See also

- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent Boolean value.

C#

```
[System.CLSCompliant(false)]
public static bool ToBoolean (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of UInt64 values to Boolean values.

C#

```
ulong[] numbers = { UInt64.MinValue, 6121, 403890774, UInt64.MaxValue };
bool result;

foreach (ulong number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-26:N0}  -->  {1}", number, result);
}
// The example displays the following output:
//       0                          -->  False
//       6,121                      -->  True
//       403,890,774                -->  True
//       18,446,744,073,709,551,615 -->  True
```

See also

- UInt64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent Boolean value.

```
C#

[System.CLSCompliant(false)]
public static bool ToBoolean (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of Int32 values to Boolean values.

```
C#

uint[] numbers = { UInt32.MinValue, 612, 4038907, Int32.MaxValue };
bool result;
```

```
foreach (uint number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-15:N0}  -->  {1}", number, result);
}
// The example displays the following output:
//       0                -->  False
//       612              -->  True
//       4,038,907        -->  True
//       2,147,483,647    -->  True
```

See also

- UInt32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to an equivalent Boolean value.

C#

```
[System.CLSCompliant(false)]
public static bool ToBoolean (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of UInt16 values to Boolean values.

C#

```
ushort[] numbers = { UInt16.MinValue, 216, 21453, UInt16.MaxValue };
bool result;

foreach (ushort number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-7:N0}  -->  {1}", number, result);
}
// The example displays the following output:
//       0        -->  False
//       216      -->  True
//       21,453   -->  True
//       65,535   -->  True
```

See also

- UInt16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(String)

Converts the specified string representation of a logical value to its Boolean equivalent.

C#

```
public static bool ToBoolean (string? value);
```

Parameters

`value` String

A string that contains the value of either TrueString or FalseString.

Returns

Boolean

`true` if `value` equals TrueString, or `false` if `value` equals FalseString or `null`.

Exceptions

FormatException

`value` is not equal to TrueString or FalseString.

Examples

The following example uses the Convert.ToBoolean(String) method to convert various strings to Boolean values.

C#

```
using System;
```

```
public class BooleanConversion
{
   public static void Main()
   {
      String[] values = { null, String.Empty, "true", "TrueString",
                          "False", "    false    ", "-1", "0" };
      foreach (var value in values) {
         try
         {
            Console.WriteLine("Converted '{0}' to {1}.", value,
                              Convert.ToBoolean(value));
         }
         catch (FormatException)
         {
            Console.WriteLine("Unable to convert '{0}' to a Boolean.",
value);
         }
      }
   }
}
// The example displays the following output:
//       Converted '' to False.
//       Unable to convert '' to a Boolean.
//       Converted 'true' to True.
//       Unable to convert 'TrueString' to a Boolean.
//       Converted 'False' to False.
//       Converted '    false    ' to False.
//       Unable to convert '-1' to a Boolean.
//       Unable to convert '0' to a Boolean.
```

Remarks

For a successful conversion to occur, the `value` parameter must equal either Boolean.TrueString, a constant whose value is `True`, Boolean.FalseString, a constant whose value is `False`, or it must be `null`. In comparing `value` with Boolean.TrueString and Boolean.FalseString, the method ignores case as well as leading and trailing white space.

If you prefer not to handle an exception if the conversion fails, you can call the Boolean.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Single)

Converts the value of the specified single-precision floating-point number to an equivalent Boolean value.

C#

```
public static bool ToBoolean (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Single values to Boolean values.

C#

```
float[] numbers = { Single.MinValue, -193.0012f, 20e-15f, 0f,
                    10551e-10f, 100.3398f, Single.MaxValue };
bool result;

foreach (float number in numbers)
```

```
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-15}  -->  {1}", number, result);
}
// The example displays the following output:
//       -3.402823E+38    -->  True
//       -193.0012        -->  True
//       2E-14            -->  True
//       0                -->  False
//       1.0551E-06       -->  True
//       100.3398         -->  True
//       3.402823E+38     -->  True
```

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Object)

Converts the value of a specified object to an equivalent Boolean value.

C#

```
public static bool ToBoolean (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Boolean

`true` or `false`, which reflects the value returned by invoking the ToBoolean(IFormatProvider) method for the underlying type of `value`. If `value` is `null`, the method returns `false`.

Exceptions

FormatException

`value` is a string that does not equal TrueString or FalseString.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion of `value` to a Boolean is not supported.

Examples

The following example converts an array of object values to Boolean values.

C#

```
object[] objects = { 16.33, -24, 0, "12", "12.7", String.Empty,
                     "1String", "True", "false", null,
                     new System.Collections.ArrayList() };

foreach (object obj in objects)
{
   Console.Write("{0,-40}  -->  ",
                 obj != null ?
                 String.Format("{0} ({1})", obj, obj.GetType().Name) :
                 "null");
   try {
      Console.WriteLine("{0}", Convert.ToBoolean(obj));
   }
   catch (FormatException) {
      Console.WriteLine("Bad Format");
   }
   catch (InvalidCastException) {
      Console.WriteLine("No Conversion");
   }
```

```
}
// The example displays the following output:
//       16.33 (Double)                              -->  True
//       -24 (Int32)                                 -->  True
//       0 (Int32)                                   -->  False
//       12 (String)                                 -->  Bad Format
//       12.7 (String)                               -->  Bad Format
//        (String)                                   -->  Bad Format
//       1String (String)                            -->  Bad Format
//       True (String)                               -->  True
//       false (String)                              -->  False
//       null                                        -->  False
//       System.Collections.ArrayList (ArrayList)    -->  No Conversion
```

See also

- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static bool ToBoolean (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Boolean

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Examples

The following example demonstrates that an attempt to convert a DateTime value to a Boolean type throws InvalidCastException.

C#

```
public void ConvertBoolean() {
    const int year          = 1979;
    const int month         = 7;
    const int day           = 28;
    const int hour          = 13;
    const int minute        = 26;
    const int second        = 15;
    const int millisecond   = 53;

    DateTime dateTime = new DateTime(year, month, day, hour,
                                     minute, second, millisecond);

    bool boolVal;

    // System.InvalidCastException is always thrown.
    try {
        boolVal = System.Convert.ToBoolean(dateTime);
    }
    catch (System.InvalidCastException) {
        System.Console.WriteLine("Conversion from DateTime to " +
            "Boolean is not supported by the .NET Framework.");
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToBoolean(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent Boolean value.

C#

```
public static bool ToBoolean (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Int32 values to Boolean values.

C#

```
int[] numbers = { Int32.MinValue, -201649, -68, 0, 612, 4038907,
                  Int32.MaxValue };
bool result;

foreach (int number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-15:N0}  -->  {1}", number, result);
}
// The example displays the following output:
```

```
//        -2,147,483,648   -->  True
//        -201,649         -->  True
//        -68              -->  True
//        0                -->  False
//        612              -->  True
//        4,038,907        -->  True
//        2,147,483,647    -->  True
```

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent Boolean value.

C#

```
public static bool ToBoolean (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Int16 values to Boolean values.

C#

```
short[] numbers = { Int16.MinValue, -10000, -154, 0, 216, 21453,
                    Int16.MaxValue };
bool result;

foreach (short number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-7:N0}  -->  {1}", number, result);
}
// The example displays the following output:
//       -32,768  -->  True
//       -10,000  -->  True
//       -154     -->  True
//       0        -->  False
//       216      -->  True
//       21,453   -->  True
//       32,767   -->  True
```

See also

- Int16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Double)

Converts the value of the specified double-precision floating-point number to an equivalent Boolean value.

```
C#

public static bool ToBoolean (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts a Boolean to a Double and a Double to a Boolean value.

```
C#

public void ConvertDoubleBool(double doubleVal) {
    bool    boolVal;
    // Double to bool conversion cannot overflow.
    boolVal = System.Convert.ToBoolean(doubleVal);
    System.Console.WriteLine("{0} as a Boolean is: {1}.",
        doubleVal, boolVal);

    // bool to double conversion cannot overflow.
    doubleVal = System.Convert.ToDouble(boolVal);
    System.Console.WriteLine("{0} as a double is: {1}.",
        boolVal, doubleVal);
}
```

See also

- Double

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Decimal)

Converts the value of the specified decimal number to an equivalent Boolean value.

C#

```
public static bool ToBoolean (decimal value);
```

Parameters

`value` Decimal

The number to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Decimal values to Boolean values.

C#

```
decimal[] numbers = { Decimal.MinValue, -12034.87m, -100m, 0m,
                              300m, 6790823.45m, Decimal.MaxValue };
bool result;
```

```
foreach (decimal number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-30}  -->  {1}", number, result);
}
// The example displays the following output:
//       -79228162514264337593543950335  -->  True
//       -12034.87                       -->  True
//       -100                            -->  True
//       0                               -->  False
//       300                             -->  True
//       6790823.45                      -->  True
//       79228162514264337593543950335   -->  True
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Char)

Calling this method always throws InvalidCastException.

C#

```
public static bool ToBoolean (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Boolean

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToBoolean(Byte)

Converts the value of the specified 8-bit unsigned integer to an equivalent Boolean value.

C#

```
public static bool ToBoolean (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Byte values to Boolean values.

C#

```
byte[] bytes = { Byte.MinValue, 100, 200, Byte.MaxValue };
bool result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToBoolean(byteValue);
   Console.WriteLine("{0,-5}  -->  {1}", byteValue, result);
}
// The example displays the following output:
//       0      -->  False
//       100    -->  True
//       200    -->  True
//       255    -->  True
```

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToBoolean(Boolean)

Returns the specified Boolean value; no actual conversion is performed.

```
C#
public static bool ToBoolean (bool value);
```

Parameters

`value` Boolean

The Boolean value to return.

Returns

Boolean

`value` is returned unchanged.

See also

- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToBoolean(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent Boolean value.

```
C#

public static bool ToBoolean (long value);
```

Parameters

value Int64

The 64-bit signed integer to convert.

Returns

Boolean

`true` if `value` is not zero; otherwise, `false`.

Examples

The following example converts an array of Int64 values to Boolean values.

```
C#

long[] numbers = { Int64.MinValue, -2016493, -689, 0, 6121,
                   403890774, Int64.MaxValue };
bool result;

foreach (long number in numbers)
{
   result = Convert.ToBoolean(number);
   Console.WriteLine("{0,-26:N0}  -->  {1}", number, result);
}
// The example displays the following output:
//       -9,223,372,036,854,775,808  -->  True
//       -2,016,493                  -->  True
//       -689                        -->  True
//       0                           -->  False
//       6,121                       -->  True
//       403,890,774                 -->  True
//       9,223,372,036,854,775,807   -->  True
```

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToByte Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to an 8-bit unsigned integer.

Overloads

Expand table

ToByte(String)	Converts the specified string representation of a number to an equivalent 8-bit unsigned integer.
ToByte(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToByte(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToByte(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 8-bit unsigned integer.
ToByte(Object, IFormatProvider)	Converts the value of the specified object to an 8-bit unsigned integer, using the specified culture-specific formatting information.
ToByte(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 8-bit unsigned integer, using specified culture-specific formatting information.
ToByte(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 8-bit unsigned integer.
ToByte(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit unsigned integer.
ToByte(SByte)	Converts the value of the specified 8-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(DateTime)	Calling this method always throws InvalidCastException.
ToByte(Int64)	Converts the value of the specified 64-bit signed integer to an

	equivalent 8-bit unsigned integer.
ToByte(Byte)	Returns the specified 8-bit unsigned integer; no actual conversion is performed.
ToByte(Char)	Converts the value of the specified Unicode character to the equivalent 8-bit unsigned integer.
ToByte(Decimal)	Converts the value of the specified decimal number to an equivalent 8-bit unsigned integer.
ToByte(Boolean)	Converts the specified Boolean value to the equivalent 8-bit unsigned integer.
ToByte(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 8-bit unsigned integer.
ToByte(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 8-bit unsigned integer.
ToByte(Object)	Converts the value of the specified object to an 8-bit unsigned integer.

ToByte(String)

Converts the specified string representation of a number to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example defines a string array and attempts to convert each string to a Byte. Note that while a `null` string parses to zero, String.Empty throws a FormatException. Also note that while leading and trailing spaces parse successfully, formatting symbols, such as currency symbols, group separators, or decimal separators, do not.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] values = { null, "", "0xC9", "C9", "101", "16.3",
                          "$12", "$12.01", "-4", "1,032", "255",
                          "   16  " };
      foreach (var value in values) {
         try {
            byte number = Convert.ToByte(value);
            Console.WriteLine("'{0}' --> {1}",
                              value == null ? "<null>" : value, number);
         }
         catch (FormatException) {
            Console.WriteLine("Bad Format: '{0}'",
                              value == null ? "<null>" : value);
         }
         catch (OverflowException) {
            Console.WriteLine("OverflowException: '{0}'", value);
         }
      }
   }
}
// The example displays the following output:
//     '<null>' --> 0
//     Bad Format: ''
//     Bad Format: '0xC9'
```

```
//      Bad Format: 'C9'
//      '101' --> 101
//      Bad Format: '16.3'
//      Bad Format: '$12'
//      Bad Format: '$12.01'
//      OverflowException: '-4'
//      Bad Format: '1,032'
//      '255' --> 255
//      '   16  ' --> 16
```

Remarks

Using the ToByte(String) method is equivalent to passing `value` to the Byte.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Byte.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(UInt16)

> Important

This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to an equivalent 8-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static byte ToByte (ushort value);
```

Parameters

value UInt16

The 16-bit unsigned integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Byte.MaxValue.

Examples

The following example converts an array of unsigned 16-bit integers to Byte values.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
byte result;
foreach (ushort number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
```

```
    }
    catch (OverflowException) {
       Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                         number.GetType().Name, number);
    }
}
// The example displays the following output:
//       Converted the UInt16 value 0 to the Byte value 0.
//       Converted the UInt16 value 121 to the Byte value 121.
//       The UInt16 value 340 is outside the range of the Byte type.
//       The UInt16 value 65535 is outside the range of the Byte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static byte ToByte (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Byte.MaxValue.

Examples

The following example converts an array of unsigned integers to Byte values.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
byte result;
foreach (uint number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       Converted the UInt32 value 0 to the Byte value 0.
//       Converted the UInt32 value 121 to the Byte value 121.
//       The UInt32 value 340 is outside the range of the Byte type.
//       The UInt32 value 4294967295 is outside the range of the Byte
type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(String, Int32)

Converts the string representation of a number in a specified base to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

Byte

An 8-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

ArgumentOutOfRangeException
`value` is Empty.

FormatException
`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException
`value`, which represents a base 10 unsigned number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example alternately attempts to interpret an array of strings as the representation of binary, octal, decimal, and hexadecimal values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int[] bases = { 2, 8, 10, 16 };
      string[] values = { "-1", "1", "08", "0F", "11" , "12", "30",
                          "101", "255", "FF", "10000000", "80" };
      byte number;
      foreach (int numBase in bases)
      {
         Console.WriteLine("Base {0}:", numBase);
         foreach (string value in values)
         {
```

```
            try {
               number = Convert.ToByte(value, numBase);
               Console.WriteLine("   Converted '{0}' to {1}.", value,
number);
            }
            catch (FormatException) {
               Console.WriteLine("   '{0}' is not in the correct format
for a base {1} byte value.",
                                 value, numBase);
            }
            catch (OverflowException) {
               Console.WriteLine("   '{0}' is outside the range of the
Byte type.", value);
            }
            catch (ArgumentException) {
               Console.WriteLine("   '{0}' is invalid in base {1}.",
value, numBase);
            }
         }
      }
   }
}
// The example displays the following output:
//    Base 2:
//       '-1' is invalid in base 2.
//       Converted '1' to 1.
//       '08' is not in the correct format for a base 2 conversion.
//       '0F' is not in the correct format for a base 2 conversion.
//       Converted '11' to 3.
//       '12' is not in the correct format for a base 2 conversion.
//       '30' is not in the correct format for a base 2 conversion.
//       Converted '101' to 5.
//       '255' is not in the correct format for a base 2 conversion.
//       'FF' is not in the correct format for a base 2 conversion.
//       Converted '10000000' to 128.
//       '80' is not in the correct format for a base 2 conversion.
//    Base 8:
//       '-1' is invalid in base 8.
//       Converted '1' to 1.
//       '08' is not in the correct format for a base 8 conversion.
//       '0F' is not in the correct format for a base 8 conversion.
//       Converted '11' to 9.
//       Converted '12' to 10.
//       Converted '30' to 24.
//       Converted '101' to 65.
//       Converted '255' to 173.
//       'FF' is not in the correct format for a base 8 conversion.
//       '10000000' is outside the range of the Byte type.
//       '80' is not in the correct format for a base 8 conversion.
//    Base 10:
//       '-1' is outside the range of the Byte type.
//       Converted '1' to 1.
//       Converted '08' to 8.
//       '0F' is not in the correct format for a base 10 conversion.
//       Converted '11' to 11.
```

```
//       Converted '12' to 12.
//       Converted '30' to 30.
//       Converted '101' to 101.
//       Converted '255' to 255.
//       'FF' is not in the correct format for a base 10 conversion.
//       '10000000' is outside the range of the Byte type.
//       Converted '80' to 80.
//    Base 16:
//       '-1' is invalid in base 16.
//       Converted '1' to 1.
//       Converted '08' to 8.
//       Converted '0F' to 15.
//       Converted '11' to 17.
//       Converted '12' to 18.
//       Converted '30' to 48.
//       '101' is outside the range of the Byte type.
//       '255' is outside the range of the Byte type.
//       Converted 'FF' to 255.
//       '10000000' is outside the range of the Byte type.
//       Converted '80' to 128.
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the Byte data type supports unsigned values only, the ToByte(String, Int32) method assumes that `value` is expressed using unsigned binary representation. In other words, all eight bits are used to represent the numeric value, and a sign bit is absent. As a result, it is possible to write code in which a signed byte value that is out of the range of the Byte data type is converted to a Byte value without the method throwing an exception. The following example converts MinValue to its hexadecimal string representation, and then calls the ToByte(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x80 converts to 128."

C#

```
// Create a hexadecimal value out of range of the Byte type.
string value = SByte.MinValue.ToString("X");
// Convert it back to a number.
try
{
   byte number = Convert.ToByte(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
{
```

```
    Console.WriteLine("Unable to convert '0x{0}' to a byte.", value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method or operator is using the appropriate numeric representation to interpret a particular value. The following example illustrates one technique for ensuring that the method does not inappropriately use unsigned binary representation when it converts a hexadecimal string representation to a Byte value. The example determines whether a value represents a signed or an unsigned integer while it is converting that value to its string representation. When the example converts the value back to a Byte value, it checks whether the original value was a signed integer. If so, and if its high-order bit is set (which indicates that the value is negative and that it uses two's complement instead of unsigned binary representation), the method throws an exception.

C#

```
// Create a negative hexadecimal value out of range of the Byte type.
sbyte sourceNumber = SByte.MinValue;
bool isSigned =
Math.Sign((sbyte)sourceNumber.GetType().GetField("MinValue").GetValue(nul
l)) == -1;
string value = sourceNumber.ToString("X");
byte targetNumber;
try
{
   targetNumber = Convert.ToByte(value, 16);
   if (isSigned && ((targetNumber & 0x80) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned byte.",
value);
}
// Displays the following to the console:
//    Unable to convert '0x80' to an unsigned byte.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Object, IFormatProvider)

Converts the value of the specified object to an 8-bit unsigned integer, using the specified culture-specific formatting information.

```
C#

public static byte ToByte (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in the property format for a Byte value.

InvalidCastException

`value` does not implement IConvertible.

-or-

Conversion from `value` to the Byte type is not supported.

OverflowException

`value` represents a number that is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example defines a `ByteString` class that implements the IConvertible interface. The class stores the string representation of a byte value along with a sign field, so that it is able to represent both signed and unsigned byte values.

C#

```
using System;
using System.Globalization;

public enum SignBit { Negative=-1, Zero=0, Positive=1 };

public struct ByteString : IConvertible
{
   private SignBit signBit;
   private string byteString;

   public SignBit Sign
   {
      set { signBit = value; }
      get { return signBit; }
   }

   public string Value
   {
      set {
         if (value.Trim().Length > 2)
            throw new ArgumentException("The string representation of a
byte cannot have more than two characters.");
         else
            byteString = value;
      }
      get { return byteString; }
   }

   // IConvertible implementations.
   public TypeCode GetTypeCode() {
      return TypeCode.Object;
   }

   public bool ToBoolean(IFormatProvider provider)
```

```
    {
        if (signBit == SignBit.Zero)
            return false;
        else
            return true;
    }

    public byte ToByte(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is out of range
of the Byte type.", Convert.ToSByte(byteString, 16)));
        else
            return Byte.Parse(byteString, NumberStyles.HexNumber);
    }

    public char ToChar(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative) {
            throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToSByte(byteString, 16)));
        }
        else {
            byte byteValue = Byte.Parse(this.byteString,
NumberStyles.HexNumber);
            return Convert.ToChar(byteValue);
        }
    }

    public DateTime ToDateTime(IFormatProvider provider)
    {
        throw new InvalidCastException("ByteString to DateTime conversion
is not supported.");
    }

    public decimal ToDecimal(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
        {
            sbyte byteValue = SByte.Parse(byteString,
NumberStyles.HexNumber);
            return Convert.ToDecimal(byteValue);
        }
        else
        {
            byte byteValue = Byte.Parse(byteString, NumberStyles.HexNumber);
            return Convert.ToDecimal(byteValue);
        }
    }

    public double ToDouble(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToDouble(SByte.Parse(byteString,
NumberStyles.HexNumber));
```

```
      else
         return Convert.ToDouble(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public short ToInt16(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToInt16(SByte.Parse(byteString,
NumberStyles.HexNumber));
      else
         return Convert.ToInt16(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public int ToInt32(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToInt32(SByte.Parse(byteString,
NumberStyles.HexNumber));
      else
         return Convert.ToInt32(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public long ToInt64(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToInt64(SByte.Parse(byteString,
NumberStyles.HexNumber));
      else
         return Convert.ToInt64(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public sbyte ToSByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         try {
            return Convert.ToSByte(Byte.Parse(byteString,
NumberStyles.HexNumber));
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",
                                                        Byte.Parse(byteString,
NumberStyles.HexNumber)), e);
         }
      else
         return SByte.Parse(byteString, NumberStyles.HexNumber);
   }

   public float ToSingle(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
```

```
            return Convert.ToSingle(SByte.Parse(byteString,
NumberStyles.HexNumber));
         else
            return Convert.ToSingle(Byte.Parse(byteString,
NumberStyles.HexNumber));
      }

      public string ToString(IFormatProvider provider)
      {
         return "0x" + this.byteString;
      }

      public object ToType(Type conversionType, IFormatProvider provider)
      {
         switch (Type.GetTypeCode(conversionType))
         {
            case TypeCode.Boolean:
               return this.ToBoolean(null);
            case TypeCode.Byte:
               return this.ToByte(null);
            case TypeCode.Char:
               return this.ToChar(null);
            case TypeCode.DateTime:
               return this.ToDateTime(null);
            case TypeCode.Decimal:
               return this.ToDecimal(null);
            case TypeCode.Double:
               return this.ToDouble(null);
            case TypeCode.Int16:
               return this.ToInt16(null);
            case TypeCode.Int32:
               return this.ToInt32(null);
            case TypeCode.Int64:
               return this.ToInt64(null);
            case TypeCode.Object:
               if (typeof(ByteString).Equals(conversionType))
                  return this;
               else
                  throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.", conversionType.Name));
            case TypeCode.SByte:
               return this.ToSByte(null);
            case TypeCode.Single:
               return this.ToSingle(null);
            case TypeCode.String:
               return this.ToString(null);
            case TypeCode.UInt16:
               return this.ToUInt16(null);
            case TypeCode.UInt32:
               return this.ToUInt32(null);
            case TypeCode.UInt64:
               return this.ToUInt64(null);
            default:
               throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
```

```
      }
   }

   public UInt16 ToUInt16(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt16 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
      else
         return Convert.ToUInt16(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public UInt32 ToUInt32(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
      else
         return Convert.ToUInt32(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public UInt64 ToUInt64(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt64 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
      else
         return Convert.ToUInt64(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }
}
```

The following example instantiates several `ByteString` objects and calls the ToByte(Object, IFormatProvider) method to convert them to byte values. It illustrates that the ToByte(Object, IFormatProvider) method wraps a call to the IConvertible.ToByte method of the object to be converted.

C#

```
public class Class1
{
   public static void Main()
   {
      byte positiveByte = 216;
```

```
        sbyte negativeByte = -101;

        ByteString positiveString = new ByteString();
        positiveString.Sign = (SignBit) Math.Sign(positiveByte);
        positiveString.Value = positiveByte.ToString("X2");

        ByteString negativeString = new ByteString();
        negativeString.Sign = (SignBit) Math.Sign(negativeByte);
        negativeString.Value = negativeByte.ToString("X2");

        try {
           Console.WriteLine("'{0}' converts to {1}.",
positiveString.Value, Convert.ToByte(positiveString));
        }
        catch (OverflowException) {
           Console.WriteLine("0x{0} is outside the range of the Byte
type.", positiveString.Value);
        }

        try {
           Console.WriteLine("'{0}' converts to {1}.",
negativeString.Value, Convert.ToByte(negativeString));
        }
        catch (OverflowException) {
           Console.WriteLine("0x{0} is outside the range of the Byte
type.", negativeString.Value);
        }
    }
}
// The example displays the following output:
//       'D8' converts to 216.
//       0x9B is outside the range of the Byte type.
```

Remarks

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent 8-bit unsigned integer, using specified culture-specific formatting information.

C#

```
public static byte ToByte (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example creates a custom NumberFormatInfo object that defines the positive sign as "pos" and the negative sign as "neg", which it uses in calls to the ToByte(String, IFormatProvider) method. It then calls the ToByte(String, IFormatProvider) method repeatedly to convert each element in a string array to a Byte value.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set several of its
      // properties that apply to unsigned bytes.
      NumberFormatInfo provider = new NumberFormatInfo();

      // These properties affect the conversion.
      provider.PositiveSign = "pos ";
      provider.NegativeSign = "neg ";

      // This property does not affect the conversion.
      // The input string cannot have a decimal separator.
      provider.NumberDecimalSeparator = ".";

      // Define an array of numeric strings.
      string[] numericStrings = { "234", "+234", "pos 234", "234.",
"255",
                                  "256", "-1" };

      foreach (string numericString in numericStrings)
      {
         Console.Write("'{0,-8}' ->   ", numericString);
         try {
            byte number = Convert.ToByte(numericString, provider);
            Console.WriteLine(number);
         }
         catch (FormatException) {
            Console.WriteLine("Incorrect Format");
         }
         catch (OverflowException) {
            Console.WriteLine("Overflows a Byte");
         }
      }
   }
}
// The example displays the following output:
//       '234     ' ->   234
```

```
//       '+234    ' ->   Incorrect Format
//       'pos 234 ' ->   234
//       '234.    ' ->   Incorrect Format
//       '255     ' ->   255
//       '256     ' ->   Overflows a Byte
//       '-1      ' ->   Incorrect Format
```

Remarks

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Byte.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Single)

Converts the value of the specified single-precision floating-point number to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (float value);
```

Parameters

`value` Single

A single-precision floating-point number.

Returns

Byte

`value`, rounded to the nearest 8-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Byte.MaxValue or less than Byte.MinValue.

Examples

The following example converts a Single value to a Byte.

C#

```
public void ConvertByteSingle(byte byteVal) {
    float floatVal;

    // Byte to float conversion will not overflow.
    floatVal = System.Convert.ToSingle(byteVal);
    System.Console.WriteLine("The byte as a float is {0}.",
        floatVal);

    // Float to byte conversion can overflow.
    try {
        byteVal = System.Convert.ToByte(floatVal);
        System.Console.WriteLine("The float as a byte is {0}.",
            byteVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "The float value is too large for a byte.");
```

```
    }
}
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static byte ToByte (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Byte.MaxValue.

Examples

The following example converts an array of unsigned long integers to Byte values.

C#

```
ulong[] numbers= { UInt64.MinValue, 121, 340, UInt64.MaxValue };
byte result;
foreach (ulong number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       Converted the UInt64 value 0 to the Byte value 0.
//       Converted the UInt64 value 121 to the Byte value 121.
//       The UInt64 value 340 is outside the range of the Byte type.
//       The UInt64 value 18446744073709551615 is outside the range of
the Byte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to an equivalent 8-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static byte ToByte (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to be converted.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than Byte.MinValue.

Examples

The following example converts an array of SByte values to Byte values.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
byte result;
foreach (sbyte number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       The SByte value -128 is outside the range of the Byte type.
//       The SByte value -1 is outside the range of the Byte type.
//       Converted the SByte value 0 to the Byte value 0.
//       Converted the SByte value 10 to the Byte value 10.
//       Converted the SByte value 127 to the Byte value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(DateTime)

Calling this method always throws InvalidCastException.

```
C#

public static byte ToByte (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Byte

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToByte(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example converts an array of Int64 values to Byte values.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
byte result;
foreach (long number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       The Int64 value -9223372036854775808 is outside the range of the
Byte type.
//       The Int64 value -1 is outside the range of the Byte type.
```

```
//       Converted the Int64 value 0 to the Byte value 0.
//       Converted the Int64 value 121 to the Byte value 121.
//       The Int64 value 340 is outside the range of the Byte type.
//       The Int64 value 9223372036854775807 is outside the range of the
Byte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Byte)

Returns the specified 8-bit unsigned integer; no actual conversion is performed.

C#

```
public static byte ToByte (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to return.

Returns

Byte

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Char)

Converts the value of the specified Unicode character to the equivalent 8-bit unsigned integer.

```
C#
public static byte ToByte (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Exceptions

OverflowException

`value` represents a number that is greater than Byte.MaxValue.

Examples

The following example converts an array of Char values to Byte values.

C#

```
char[] chars = { 'a', 'z', '\x0007', '\x03FF' };
foreach (char ch in chars)
{
   try {
      byte result = Convert.ToByte(ch);
      Console.WriteLine("{0} is converted to {1}.", ch, result);
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to convert u+{0} to a byte.",
                        Convert.ToInt16(ch).ToString("X4"));
   }
}
// The example displays the following output:
//       a is converted to 97.
//       z is converted to 122.
//        is converted to 7.
//       Unable to convert u+03FF to a byte.
```

Remarks

This method returns an unsigned byte value that represents the numeric code of the Char object passed to it. In .NET, a Char object is a 16-bit value. This means that the method is suitable for returning the numeric codes of characters in the ASCII character range or in the Unicode C0 Controls and Basic Latin, and C1 Controls and Latin-1 Supplement ranges, from U+0000 to U+00FF.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Decimal)

Converts the value of the specified decimal number to an equivalent 8-bit unsigned integer.

```C#
public static byte ToByte (decimal value);
```

Parameters

`value` Decimal

The number to convert.

Returns

Byte

`value`, rounded to the nearest 8-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Byte.MaxValue or less than Byte.MinValue.

Remarks

The following example converts a Byte value to a Decimal and a Decimal value to a Byte.

```C#
public void ConvertByteDecimal(byte byteVal) {
    decimal decimalVal;

    // Byte to decimal conversion will not overflow.
    decimalVal = System.Convert.ToDecimal(byteVal);
    System.Console.WriteLine("The byte as a decimal is {0}.",
        decimalVal);

    // Decimal to byte conversion can overflow.
    try {
        byteVal = System.Convert.ToByte(decimalVal);
        System.Console.WriteLine("The Decimal as a byte is {0}.",
            byteVal);
```

```
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "The decimal value is too large for a byte.");
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Boolean)

Converts the specified Boolean value to the equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Byte

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example illustrates the conversion of Boolean to Byte values.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToByte(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToByte(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example converts an array of Int16 values to Byte values.

C#

```
short[] numbers = { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
byte result;
foreach (short number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       The Int16 value -32768 is outside the range of the Byte type.
//       The Int16 value -1 is outside the range of the Byte type.
//       Converted the Int16 value 0 to the Byte value 0.
//       Converted the Int16 value 121 to the Byte value 121.
//       The Int16 value 340 is outside the range of the Byte type.
//       The Int16 value 32767 is outside the range of the Byte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example converts an array of Int32 values to Byte values.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
byte result;
foreach (int number in numbers)
{
   try {
      result = Convert.ToByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//       The Int32 value -2147483648 is outside the range of the Byte
type.
//       The Int32 value -1 is outside the range of the Byte type.
//       Converted the Int32 value 0 to the Byte value 0.
//       Converted the Int32 value 121 to the Byte value 121.
//       The Int32 value 340 is outside the range of the Byte type.
//       The Int32 value 2147483647 is outside the range of the Byte
type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Double)

Converts the value of the specified double-precision floating-point number to an equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Byte

`value`, rounded to the nearest 8-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Byte.MaxValue or less than Byte.MinValue.

Examples

The following example converts a Byte value to a Double and a Double value to a Byte.

C#

```
public void ConvertDoubleByte(double doubleVal) {
    byte    byteVal = 0;

    // Double to byte conversion can overflow.
    try {
        byteVal = System.Convert.ToByte(doubleVal);
        System.Console.WriteLine("{0} as a byte is: {1}.",
            doubleVal, byteVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in double-to-byte conversion.");
    }

    // Byte to double conversion cannot overflow.
    doubleVal = System.Convert.ToDouble(byteVal);
    System.Console.WriteLine("{0} as a double is: {1}.",
```

```
        byteVal, doubleVal);
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByte(Object)

Converts the value of the specified object to an 8-bit unsigned integer.

C#

```
public static byte ToByte (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Byte

An 8-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in the property format for a Byte value.

InvalidCastException

`value` does not implement IConvertible.

-or-

Conversion from `value` to the Byte type is not supported.

OverflowException

`value` represents a number that is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example uses the ToByte(Object) method to convert an array of objects to Byte values.

C#

```
object[] values = { true, -12, 163, 935, 'x', "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2};
byte result;
foreach (object value in values)
{
   try {
      result = Convert.ToByte(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("The {0} value {1} is outside the range of the
Byte type.",
                        value.GetType().Name, value);
   }
   catch (FormatException)
   {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException)
   {
      Console.WriteLine("No conversion to a Byte exists for the {0} value
{1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
```

```
//       Converted the Boolean value True to the Byte value 1.
//       The Int32 value -12 is outside the range of the Byte type.
//       Converted the Int32 value 163 to the Byte value 163.
//       The Int32 value 935 is outside the range of the Byte type.
//       Converted the Char value x to the Byte value 120.
//       Converted the String value 104 to the Byte value 104.
//       The String value 103.0 is not in a recognizable format.
//       The String value -1 is outside the range of the Byte type.
//       The String value 1.00e2 is not in a recognizable format.
//       The String value One is not in a recognizable format.
//       Converted the Double value 100 to the Byte value 100.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToByte implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToChar Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a Unicode character.

Overloads

Expand table

ToChar(String, IFormatProvider)	Converts the first character of a specified string to a Unicode character, using specified culture-specific formatting information.
ToChar(Object)	Converts the value of the specified object to a Unicode character.
ToChar(Object, IFormatProvider)	Converts the value of the specified object to its equivalent Unicode character, using the specified culture-specific formatting information.
ToChar(UInt64)	Converts the value of the specified 64-bit unsigned integer to its equivalent Unicode character.
ToChar(UInt32)	Converts the value of the specified 32-bit unsigned integer to its equivalent Unicode character.
ToChar(UInt16)	Converts the value of the specified 16-bit unsigned integer to its equivalent Unicode character.
ToChar(String)	Converts the first character of a specified string to a Unicode character.
ToChar(Single)	Calling this method always throws InvalidCastException.
ToChar(SByte)	Converts the value of the specified 8-bit signed integer to its equivalent Unicode character.
ToChar(Int64)	Converts the value of the specified 64-bit signed integer to its equivalent Unicode character.
ToChar(Int16)	Converts the value of the specified 16-bit signed integer to its equivalent Unicode character.
ToChar(Double)	Calling this method always throws InvalidCastException.

ToChar(Decimal)	Calling this method always throws InvalidCastException.
ToChar(DateTime)	Calling this method always throws InvalidCastException.
ToChar(Char)	Returns the specified Unicode character value; no actual conversion is performed.
ToChar(Byte)	Converts the value of the specified 8-bit unsigned integer to its equivalent Unicode character.
ToChar(Boolean)	Calling this method always throws InvalidCastException.
ToChar(Int32)	Converts the value of the specified 32-bit signed integer to its equivalent Unicode character.

ToChar(String, IFormatProvider)

Converts the first character of a specified string to a Unicode character, using specified culture-specific formatting information.

C#

```
public static char ToChar (string value, IFormatProvider? provider);
```

Parameters

`value` String

A string of length 1 or `null`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information. This parameter is ignored.

Returns

Char

A Unicode character that is equivalent to the first and only character in `value`.

Exceptions

ArgumentNullException

`value` is `null`.

FormatException

The length of `value` is not 1.

Examples

The following example converts a string representation of a Char value with the `ToChar` method, using an IFormatProvider object that displays the type of the format provider for which it is called. The example shows that the format provider is not referenced.

C#

```
using System;
using System.Globalization;

public class DummyProvider : IFormatProvider
{
    // Normally, GetFormat returns an object of the requested type
    // (usually itself) if it is able; otherwise, it returns Nothing.
    public object GetFormat(Type argType)
    {
        // Here, GetFormat displays the name of argType, after removing
        // the namespace information. GetFormat always returns null.
        string argStr = argType.ToString( );
        if( argStr == "" )
            argStr = "Empty";
        argStr = argStr.Substring( argStr.LastIndexOf( '.' ) + 1 );

        Console.Write( "{0,-20}", argStr );
        return null;
    }
}

class ConvertNonNumericProviderDemo
{
    public static void Main( )
    {
        // Create an instance of IFormatProvider.
        DummyProvider provider = new DummyProvider( );
        string format   = "{0,-17}{1,-17}{2}";

        // Convert these values using DummyProvider.
        string Int32A   = "-252645135";
        string DoubleA  = "61680.3855";
        string DayTimeA = "2001/9/11 13:45";

        string BoolA    = "True";
        string StringA  = "Qwerty";
        string CharA    = "$";

        Console.WriteLine( "This example of selected " +
```

```
            "Convert.To<Type>( String, IFormatProvider ) \nmethods " +
            "generates the following output. The example displays " +
            "the \nprovider type if the IFormatProvider is called." );
        Console.WriteLine( "\nNote: For the " +
            "ToBoolean, ToString, and ToChar methods, the \n" +
            "IFormatProvider object is not referenced." );

        // The format provider is called for the following conversions.
        Console.WriteLine( );
        Console.WriteLine( format, "ToInt32", Int32A,
            Convert.ToInt32( Int32A, provider ) );
        Console.WriteLine( format, "ToDouble", DoubleA,
            Convert.ToDouble( DoubleA, provider ) );
        Console.WriteLine( format, "ToDateTime", DayTimeA,
            Convert.ToDateTime( DayTimeA, provider ) );

        // The format provider is not called for these conversions.
        Console.WriteLine( );
        Console.WriteLine( format, "ToBoolean", BoolA,
            Convert.ToBoolean( BoolA, provider ) );
        Console.WriteLine( format, "ToString", StringA,
            Convert.ToString( StringA, provider ) );
        Console.WriteLine( format, "ToChar", CharA,
            Convert.ToChar( CharA, provider ) );
    }
}

/*
This example of selected Convert.To<Type>( String, IFormatProvider )
methods generates the following output. The example displays the
provider type if the IFormatProvider is called.

Note: For the ToBoolean, ToString, and ToChar methods, the
IFormatProvider object is not referenced.

NumberFormatInfo    ToInt32          -252645135       -252645135
NumberFormatInfo    ToDouble         61680.3855       61680.3855
DateTimeFormatInfo  ToDateTime       2001/9/11 13:45  9/11/2001 1:45:00
PM

ToBoolean        True             True
ToString         Qwerty           Qwerty
ToChar           $                $
*/
```

Remarks

`value` must be a string that contains a single character.

If you prefer not to handle an exception if the conversion fails, you can call the Char.TryParse method instead. It returns a Boolean value that indicates whether the

conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Object)

Converts the value of the specified object to a Unicode character.

C#

```
public static char ToChar (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

Returns

Char

A Unicode character that is equivalent to value, or Char.MinValue if `value` is `null`.

Exceptions

ArgumentNullException

`value` is a null string.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion of `value` to a Char is not supported.

OverflowException

`value` is less than Char.MinValue or greater than Char.MaxValue.

Examples

The following example attempts to convert each element in an object array to a Char value.

C#

```
object[] values = { 'r', "s", "word", (byte) 83, 77, 109324, 335812911,
                    new DateTime(2009, 3, 10), (uint) 1934,
                    (sbyte) -17, 169.34, 175.6m, null };
char result;

foreach (object value in values)
{
   try {
      result = Convert.ToChar(value);
      Console.WriteLine("The {0} value {1} converts to {2}.",
                        value.GetType().Name, value, result);
   }
   catch (FormatException e) {
      Console.WriteLine(e.Message);
   }
   catch (InvalidCastException) {
      Console.WriteLine("Conversion of the {0} value {1} to a Char is not
supported.",
                        value.GetType().Name, value);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Char data type.",
                        value.GetType().Name, value);
   }
   catch (NullReferenceException) {
      Console.WriteLine("Cannot convert a null reference to a Char.");
   }
}
// The example displays the following output:
//       The Char value r converts to r.
//       The String value s converts to s.
//       String must be exactly one character long.
//       The Byte value 83 converts to S.
```

```
//       The Int32 value 77 converts to M.
//       The Int32 value 109324 is outside the range of the Char data
type.
//       The Int32 value 335812911 is outside the range of the Char data
type.
//       Conversion of the DateTime value 3/10/2009 12:00:00 AM to a Char
is not supported.
//       The UInt32 value 1934 converts to ?.
//       The SByte value -17 is outside the range of the Char data type.
//       Conversion of the Double value 169.34 to a Char is not
supported.
//       Conversion of the Decimal value 175.6 to a Char is not
supported.
//       Cannot convert a null reference to a Char.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToChar implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Object, IFormatProvider)

Converts the value of the specified object to its equivalent Unicode character, using the specified culture-specific formatting information.

C#

```
public static char ToChar (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Char

A Unicode character that is equivalent to `value`, or Char.MinValue if `value` is `null`.

Exceptions

ArgumentNullException

`value` is a null string.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion of `value` to a Char is not supported.

OverflowException

`value` is less than Char.MinValue or greater than Char.MaxValue.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the `ToChar` method. This method returns a character whose Unicode value is the average of the array of Double values, using the object that implements IFormatProvider to determine how to calculate the average.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
```

```
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }

    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;

    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
```

```
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
        {
            if( index >= 0 && index < data.Count )
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;

        if( data.Count == 0 )
            return 0.0;

        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];
```

```
                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];

            default:
                return 0.0;
        }
    }

    // Get the AverageInfo object from the caller's format provider,
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }
```

```
    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
            GetAverageInfo( provider ).TypeOfAverage )
        {
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
            Byte.MinValue, Byte.MaxValue, provider ) );
    }

    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
```

```
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
    }

    public double ToDouble( IFormatProvider provider )
    {
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
```

```
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }

    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}

class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );
```

```
 // Display the dataset elements.
if( ds.Count > 0 )
{
    Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
    for( int iX = 1; iX < ds.Count; iX++ )
        Console.Write( ", {0}", ds[ iX ] );
    Console.WriteLine( "]\n" );
}

Console.WriteLine( fmt, "Convert.", "Default",
    "Geometric Mean", "Median");
Console.WriteLine( fmt, "--------", "-------",
    "--------------", "------");
Console.WriteLine( fmt, "ToBoolean",
    Convert.ToBoolean( ds, null ),
    Convert.ToBoolean( ds, geMean ),
    Convert.ToBoolean( ds, median ) );
Console.WriteLine( fmt, "ToByte",
    Convert.ToByte( ds, null ),
    Convert.ToByte( ds, geMean ),
    Convert.ToByte( ds, median ) );
Console.WriteLine( fmt, "ToChar",
    Convert.ToChar( ds, null ),
    Convert.ToChar( ds, geMean ),
    Convert.ToChar( ds, median ) );
Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
    "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
    "ToDateTime", Convert.ToDateTime( ds, null ),
    Convert.ToDateTime( ds, geMean ),
    Convert.ToDateTime( ds, median ) );
Console.WriteLine( fmt, "ToDecimal",
    Convert.ToDecimal( ds, null ),
    Convert.ToDecimal( ds, geMean ),
    Convert.ToDecimal( ds, median ) );
Console.WriteLine( fmt, "ToDouble",
    Convert.ToDouble( ds, null ),
    Convert.ToDouble( ds, geMean ),
    Convert.ToDouble( ds, median ) );
Console.WriteLine( fmt, "ToInt16",
    Convert.ToInt16( ds, null ),
    Convert.ToInt16( ds, geMean ),
    Convert.ToInt16( ds, median ) );
Console.WriteLine( fmt, "ToInt32",
    Convert.ToInt32( ds, null ),
    Convert.ToInt32( ds, geMean ),
    Convert.ToInt32( ds, median ) );
Console.WriteLine( fmt, "ToInt64",
    Convert.ToInt64( ds, null ),
    Convert.ToInt64( ds, geMean ),
    Convert.ToInt64( ds, median ) );
Console.WriteLine( fmt, "ToSByte",
    Convert.ToSByte( ds, null ),
    Convert.ToSByte( ds, geMean ),
    Convert.ToSByte( ds, median ) );
Console.WriteLine( fmt, "ToSingle",
```

```
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );

        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );

        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       103                  59                  67
ToChar                         g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                      103                  59                  67
ToInt32                      103                  59                  67
ToInt64                      103                  59                  67
ToSByte                      103                  59                  67
ToSingle                 103.4917            59.43321                67.3
```

```
ToUInt16                    103                 59                  67
ToUInt32                    103                 59                  67
ToUInt64                    103                 59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal              750999.89    631577.237188435            499999.5
ToDouble               750999.89    631577.237188435            499999.5
ToInt16                    32767               32767               32767
ToInt32                   751000              631577              500000
ToInt64                   751000              631577              500000
ToSByte                      127                 127                 127
ToSingle                750999.9            631577.3            499999.5
ToUInt16                   65535               65535               65535
ToUInt32                  751000              631577              500000
ToUInt64                  751000              631577              500000
*/
```

Remarks

The return value is the result of invoking the IConvertible.ToChar method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to its equivalent Unicode character.

```
C#

[System.CLSCompliant(false)]
public static char ToChar (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Char.MaxValue.

Examples

The following example converts each element in an array of unsigned long integers to a Char value.

```
C#

ulong[] numbers = { UInt64.MinValue, 40, 160, 255, 1028,
                    2011, 30001, 207154, Int64.MaxValue };
char result;
```

```
foreach (ulong number in numbers)
{
   try {
      result = Convert.ToChar(number);
      Console.WriteLine("{0} converts to '{1}'.", number, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Char data
type.",
                        number);
   }
}
// The example displays the following output:
//       0 converts to ' '.
//       40 converts to '('.
//       160 converts to ' '.
//       255 converts to 'ÿ'.
//       1028 converts to 'Є'.
//       2011 converts to 'ۂ'.
//       30001 converts to '由'.
//       207154 is outside the range of the Char data type.
//       9223372036854775807 is outside the range of the Char data type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to its equivalent Unicode character.

```
C#

[System.CLSCompliant(false)]
public static char ToChar (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Char.MaxValue.

Examples

The following example converts each element in an array of unsigned integers to a Char value.

```
C#

uint[] numbers = { UInt32.MinValue, 40, 160, 255, 1028,
                   2011, 30001, 207154, Int32.MaxValue };
char result;
foreach (uint number in numbers)
{
   try {
      result = Convert.ToChar(number);
      Console.WriteLine("{0} converts to '{1}'.", number, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Char data
type.",
```

```
                                number);
    }
}
// The example displays the following output:
//       0 converts to ' '.
//       40 converts to '('.
//       160 converts to ' '.
//       255 converts to 'ÿ'.
//       1028 converts to 'Є'.
//       2011 converts to 'ϖ'.
//       30001 converts to '由'.
//       207154 is outside the range of the Char data type.
//       2147483647 is outside the range of the Char data type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to its equivalent Unicode character.

C#

```
[System.CLSCompliant(false)]
public static char ToChar (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned 16-bit integers to a Char value.

C#

```
ushort[] numbers = { UInt16.MinValue, 40, 160, 255, 1028,
                     2011, UInt16.MaxValue };
char result;
foreach (ushort number in numbers)
{
   result = Convert.ToChar(number);
   Console.WriteLine("{0} converts to '{1}'.", number, result);
}
// The example displays the following output:
//       0 converts to ' '.
//       40 converts to '('.
//       160 converts to ' '.
//       255 converts to 'ÿ'.
//       1028 converts to 'Є'.
//       2011 converts to 'ϖ'.
//       65535 converts to '￿'.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(String)

Converts the first character of a specified string to a Unicode character.

```
C#
public static char ToChar (string value);
```

Parameters

`value` String

A string of length 1.

Returns

Char

A Unicode character that is equivalent to the first and only character in `value`.

Exceptions

ArgumentNullException

`value` is `null`.

FormatException

The length of `value` is not 1.

Examples

The following example converts each element in a string array to a Char value.

```
C#
```

```
string nullString = null;
string[] strings = { "A", "This",  '\u0007'.ToString(), nullString };
char result;
foreach (string strng in strings)
{
   try {
      result = Convert.ToChar(strng);
      Console.WriteLine("'{0}' converts to '{1}'.", strng, result);
   }
   catch (FormatException)
   {
      Console.WriteLine("'{0}' is not in the correct format for
conversion to a Char.",
                        strng);
   }
   catch (ArgumentNullException) {
      Console.WriteLine("A null string cannot be converted to a Char.");
   }
}
// The example displays the following output:
//       'A' converts to 'A'.
//       'This' is not in the correct format for conversion to a Char.
//       '       ' converts to ' '.
//       A null string cannot be converted to a Char.
```

Remarks

`value` must be a string that contains a single character.

If you prefer not to handle an exception if the conversion fails, you can call the Char.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Single)

Calling this method always throws InvalidCastException.

C#

```
public static char ToChar (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Char

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(SByte)

Important

This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to its equivalent Unicode character.

C#

```
[System.CLSCompliant(false)]
public static char ToChar (sbyte value);
```

Parameters

value SByte

The 8-bit signed integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than Char.MinValue.

Examples

The following example converts an array of signed bytes to Char values.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 40, 80, 120, SByte.MaxValue };
char result;
foreach (sbyte number in numbers)
{
   try {
      result = Convert.ToChar(number);
      Console.WriteLine("{0} converts to '{1}'.", number, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Char data
```

```
type.",
                                number);
    }
}
// The example displays the following output:
//       -128 is outside the range of the Char data type.
//       -1 is outside the range of the Char data type.
//       40 converts to '('.
//       80 converts to 'P'.
//       120 converts to 'x'.
//       127 converts to '⌂'.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Int64)

Converts the value of the specified 64-bit signed integer to its equivalent Unicode character.

C#

```
public static char ToChar (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Char.MinValue or greater than Char.MaxValue.

Examples

The following example attempts to convert a long integer to a Char, and throws a OverflowException on failure.

C#

```
public void ConvertLongChar(long longVal) {

    char    charVal = 'a';

    try {
        charVal = System.Convert.ToChar(longVal);
        System.Console.WriteLine("{0} as a char is {1}",
            longVal, charVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in long-to-char conversion.");
    }

    // A conversion from Char to long cannot overflow.
    longVal = System.Convert.ToInt64(charVal);
    System.Console.WriteLine("{0} as an Int64 is {1}",
        charVal, longVal);
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToChar(Int16)

Converts the value of the specified 16-bit signed integer to its equivalent Unicode character.

```
C#

public static char ToChar (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Char.MinValue.

Examples

The following example converts an array of signed 16-bit integers to Char values.

```
C#

short[] numbers = { Int16.MinValue, 0, 40, 160, 255, 1028,
                    2011, Int16.MaxValue };
char result;
foreach (short number in numbers)
{
   try {
      result = Convert.ToChar(number);
      Console.WriteLine("{0} converts to '{1}'.", number, result);
```

```
    }
    catch (OverflowException) {
       Console.WriteLine("{0} is outside the range of the Char data
type.",
                         number);
    }
}
// The example displays the following output:
//       -32768 is outside the range of the Char data type.
//       0 converts to ' '.
//       40 converts to '('.
//       160 converts to ' '.
//       255 converts to 'ÿ'.
//       1028 converts to 'Є'.
//       2011 converts to 'ˆ'.
//       32767 converts to '翻'.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Double)

Calling this method always throws InvalidCastException.

C#

```
public static char ToChar (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Char

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(Decimal)

Calling this method always throws InvalidCastException.

C#

```
public static char ToChar (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Char

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Examples

The following example attempts to convert a Decimal to Char, and throws InvalidCastException on failure.

C#

```
public void ConvertCharDecimal(char charVal) {
    Decimal decimalVal = 0;

    // Char to decimal conversion is not supported and will always
    // throw an InvalidCastException.
    try {
        decimalVal = System.Convert.ToDecimal(charVal);
    }
    catch (System.InvalidCastException) {
        System.Console.WriteLine(
            "Char-to-Decimal conversion is not supported " +
            "by the .NET Framework.");
    }

    //Decimal to char conversion is also not supported.
    try {
        charVal = System.Convert.ToChar(decimalVal);
    }
    catch (System.InvalidCastException) {
        System.Console.WriteLine(
            "Decimal-to-Char conversion is not supported " +
            "by the .NET Framework.");
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(DateTime)

Calling this method always throws InvalidCastException.

```
C#

public static char ToChar (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Char

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(Char)

Returns the specified Unicode character value; no actual conversion is performed.

```
C#
```

```
public static char ToChar (char value);
```

Parameters

`value` Char

The Unicode character to return.

Returns

Char

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(Byte)

Converts the value of the specified 8-bit unsigned integer to its equivalent Unicode character.

C#

```
public static char ToChar (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Examples

The following example converts an array of unsigned bytes to Char values.

C#

```
byte[] bytes = {Byte.MinValue, 40, 80, 120, 180, Byte.MaxValue};
char result;
foreach (byte number in bytes)
{
   result = Convert.ToChar(number);
   Console.WriteLine("{0} converts to '{1}'.", number, result);
}
// The example displays the following output:
//       0 converts to ' '.
//       40 converts to '('.
//       80 converts to 'P'.
//       120 converts to 'x'.
//       180 converts to '''.
//       255 converts to 'ÿ'.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToChar(Boolean)

Calling this method always throws InvalidCastException.

C#

```
public static char ToChar (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Char

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToChar(Int32)

Converts the value of the specified 32-bit signed integer to its equivalent Unicode character.

C#

```
public static char ToChar (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Char

A Unicode character that is equivalent to `value`.

Exceptions

OverflowException

`value` is less than Char.MinValue or greater than Char.MaxValue.

Examples

The following example converts an array of signed integers to Char values.

C#

```
int[] numbers = { -1, 0, 40, 160, 255, 1028,
                  2011, 30001, 207154, Int32.MaxValue };
char result;
foreach (int number in numbers)
{
   try {
      result = Convert.ToChar(number);
      Console.WriteLine("{0} converts to '{1}'.", number, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Char data
type.",
                        number);
   }
}
//       -1 is outside the range of the Char data type.
//       0 converts to ' '.
//       40 converts to '('.
//       160 converts to ' '.
//       255 converts to 'ÿ'.
//       1028 converts to 'Є'.
//       2011 converts to 'ϖ'.
//       30001 converts to '由'.
//       207154 is outside the range of the Char data type.
//       2147483647 is outside the range of the Char data type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToDateTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a DateTime value.

Overloads

Expand table

ToDateTime(Single)	Calling this method always throws InvalidCastException.
ToDateTime(String)	Converts the specified string representation of a date and time to an equivalent date and time value.
ToDateTime(UInt16)	Calling this method always throws InvalidCastException.
ToDateTime(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent date and time, using the specified culture-specific formatting information.
ToDateTime(UInt64)	Calling this method always throws InvalidCastException.
ToDateTime(Object, IFormatProvider)	Converts the value of the specified object to a DateTime object, using the specified culture-specific formatting information.
ToDateTime(SByte)	Calling this method always throws InvalidCastException.
ToDateTime(UInt32)	Calling this method always throws InvalidCastException.
ToDateTime(Object)	Converts the value of the specified object to a DateTime object.
ToDateTime(Int16)	Calling this method always throws InvalidCastException.
ToDateTime(Int32)	Calling this method always throws InvalidCastException.
ToDateTime(Int64)	Calling this method always throws InvalidCastException.
ToDateTime(Double)	Calling this method always throws InvalidCastException.
ToDateTime(Decimal)	Calling this method always throws InvalidCastException.

ToDateTime(DateTime)	Returns the specified DateTime object; no actual conversion is performed.
ToDateTime(Char)	Calling this method always throws InvalidCastException.
ToDateTime(Byte)	Calling this method always throws InvalidCastException.
ToDateTime(Boolean)	Calling this method always throws InvalidCastException.

ToDateTime(Single)

Calling this method always throws InvalidCastException.

```
C#
public static DateTime ToDateTime (float value);
```

Parameters

`value` Single

The single-precision floating-point value to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(String)

Converts the specified string representation of a date and time to an equivalent date and time value.

C#

```
public static DateTime ToDateTime (string? value);
```

Parameters

`value` String

The string representation of a date and time.

Returns

DateTime

The date and time equivalent of the value of `value`, or the date and time equivalent of DateTime.MinValue if `value` is `null`.

Exceptions

FormatException

`value` is not a properly formatted date and time string.

Examples

The following example uses the ToDateTime method to convert various string representations of dates and times to DateTime values.

C#

```
using System;

public class ConversionToDateTime
{
   public static void Main()
   {
      string dateString = null;

      // Convert a null string.
      ConvertToDateTime(dateString);

      // Convert an empty string.
      dateString = String.Empty;
      ConvertToDateTime(dateString);

      // Convert a non-date string.
      dateString = "not a date";
      ConvertToDateTime(dateString);

      // Try to convert various date strings.
      dateString = "05/01/1996";
      ConvertToDateTime(dateString);
      dateString = "Tue Apr 28, 2009";
      ConvertToDateTime(dateString);
      dateString = "Wed Apr 28, 2009";
      ConvertToDateTime(dateString);
      dateString = "06 July 2008 7:32:47 AM";
      ConvertToDateTime(dateString);
      dateString = "17:32:47.003";
      ConvertToDateTime(dateString);
      // Convert a string returned by DateTime.ToString("R").
      dateString = "Sat, 10 May 2008 14:32:17 GMT";
      ConvertToDateTime(dateString);
      // Convert a string returned by DateTime.ToString("o").
      dateString = "2009-05-01T07:54:59.9843750-04:00";
      ConvertToDateTime(dateString);
   }

   private static void ConvertToDateTime(string value)
   {
      DateTime convertedDate;
      try {
         convertedDate = Convert.ToDateTime(value);
         Console.WriteLine("'{0}' converts to {1} {2} time.",
                           value, convertedDate,
                           convertedDate.Kind.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in the proper format.", value);
      }
   }
}
// The example displays the following output:
//    '' converts to 1/1/0001 12:00:00 AM Unspecified time.
```

```
//    '' is not in the proper format.
//    'not a date' is not in the proper format.
//    '05/01/1996' converts to 5/1/1996 12:00:00 AM Unspecified time.
//    'Tue Apr 28, 2009' converts to 4/28/2009 12:00:00 AM Unspecified
time.
//    'Wed Apr 28, 2009' is not in the proper format.
//    '06 July 2008 7:32:47 AM' converts to 7/6/2008 7:32:47 AM
Unspecified time.
//    '17:32:47.003' converts to 5/30/2008 5:32:47 PM Unspecified time.
//    'Sat, 10 May 2008 14:32:17 GMT' converts to 5/10/2008 7:32:17 AM
Local time.
//    '2009-05-01T07:54:59.9843750-04:00' converts to 5/1/2009 4:54:59 AM
Local time.
```

Remarks

If `value` is not `null`, the return value is the result of invoking the DateTime.Parse method on `value` using the formatting information in a DateTimeFormatInfo object that is initialized for the current culture. The `value` argument must contain the representation of a date and time in one of the formats described in the DateTimeFormatInfo topic. If `value` is `null`, the method returns DateTime.MinValue.

This method tries to parse `value` completely and avoid throwing a FormatException. It completes missing month, day, and year information with the current date. If `value` contains only a date and no time, this method assumes a time of midnight. Any leading, inner, or trailing white-space characters in `value` are ignored.

If you prefer not to handle an exception if the conversion fails, you can call the DateTime.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- CurrentCulture
- Parsing Date and Time Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDateTime(UInt16)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

C#

```
[System.CLSCompliant(false)]
public static DateTime ToDateTime (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

See also

- UInt16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent date and time, using the specified culture-specific formatting information.

C#

```
public static DateTime ToDateTime (string? value, IFormatProvider?
provider);
```

Parameters

`value` String

A string that contains a date and time to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

DateTime

The date and time equivalent of the value of `value`, or the date and time equivalent of DateTime.MinValue if `value` is `null`.

Exceptions

FormatException

`value` is not a properly formatted date and time string.

Examples

The following example converts string representations of date values with the `ToDateTime` method, using an IFormatProvider object.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("{0,-18}{1,-12}{2}\n", "Date String", "Culture",
"Result");

      string[] cultureNames = { "en-US", "ru-RU","ja-JP" };
      string[] dateStrings = { "01/02/09", "2009/02/03",  "01/2009/03",
                               "01/02/2009", "21/02/09", "01/22/09",
                               "01/02/23" };
      // Iterate each culture name in the array.
      foreach (string cultureName in cultureNames)
      {
         CultureInfo culture = new CultureInfo(cultureName);

         // Parse each date using the designated culture.
         foreach (string dateStr in dateStrings)
         {
            DateTime dateTimeValue;
            try {
               dateTimeValue = Convert.ToDateTime(dateStr, culture);
                // Display the date and time in a fixed format.
                Console.WriteLine("{0,-18}{1,-12}{2:yyyy-MMM-dd}",
                                  dateStr, cultureName, dateTimeValue);
            }
            catch (FormatException e) {
                Console.WriteLine("{0,-18}{1,-12}{2}",
                                  dateStr, cultureName,
e.GetType().Name);
            }
         }
         Console.WriteLine();
      }
   }
}
```

Remarks

The return value is the result of invoking the DateTime.Parse(String, IFormatProvider) method on `value`.

`provider` is an IFormatProvider instance that obtains a DateTimeFormatInfo object. The DateTimeFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the DateTimeFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the DateTime.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parsing Date and Time Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDateTime(UInt64)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

```
C#
```

```
[System.CLSCompliant(false)]
public static DateTime ToDateTime (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

See also

- UInt64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Object, IFormatProvider)

Converts the value of the specified object to a DateTime object, using the specified culture-specific formatting information.

```
C#

public static DateTime ToDateTime (object? value, IFormatProvider?
provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

DateTime

The date and time equivalent of the value of `value`, or the date and time equivalent of DateTime.MinValue if `value` is `null`.

Exceptions

FormatException

`value` is not a valid date and time value.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

Examples

The following example defines a custom format provider, `CustomProvider`, whose GetFormat method outputs a message to the console that it has been invoked, and then returns the DateTimeFormatInfo object of the culture whose name was passed as a parameter to its class constructor. Each of these `CustomProvider` objects is used to convert the elements in an object array to date and time values. The output

indicates that the `CustomProvider` object is used in the conversion only when the type of the `value` parameter is a String.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] cultureNames = { "en-US", "hu-HU", "pt-PT" };
      object[] objects = { 12, 17.2, false, new DateTime(2010, 1, 1),
"today",
                           new System.Collections.ArrayList(), 'c',
                           "05/10/2009 6:13:18 PM", "September 8, 1899"
};

      foreach (string cultureName in cultureNames)
      {
         Console.WriteLine("{0} culture:", cultureName);
         CustomProvider provider = new CustomProvider(cultureName);
         foreach (object obj in objects)
         {
            try {
               DateTime dateValue = Convert.ToDateTime(obj, provider);
               Console.WriteLine("{0} --> {1}", obj,
                                 dateValue.ToString(new
CultureInfo(cultureName)));
            }
            catch (FormatException) {
               Console.WriteLine("{0} --> Bad Format", obj);
            }
            catch (InvalidCastException) {
               Console.WriteLine("{0} --> Conversion Not Supported",
obj);
            }
         }
         Console.WriteLine();
      }
   }
}

public class CustomProvider : IFormatProvider
{
   private string cultureName;

   public CustomProvider(string cultureName)
   {
      this.cultureName = cultureName;
   }
```

```
    public object GetFormat(Type formatType)
    {
        if (formatType == typeof(DateTimeFormatInfo))
        {
            Console.Write("(CustomProvider retrieved.) ");
            return new CultureInfo(cultureName).GetFormat(formatType);
        }
        else
        {
            return null;
        }
    }
}
// The example displays the following output:
//    en-US culture:
//    12 --> Conversion Not Supported
//    17.2 --> Conversion Not Supported
//    False --> Conversion Not Supported
//    1/1/2010 12:00:00 AM --> 1/1/2010 12:00:00 AM
//    (CustomProvider retrieved.) today --> Bad Format
//    System.Collections.ArrayList --> Conversion Not Supported
//    c --> Conversion Not Supported
//    (CustomProvider retrieved.) 05/10/2009 6:13:18 PM --> 5/10/2009
6:13:18 PM
//    (CustomProvider retrieved.) September 8, 1899 --> 9/8/1899 12:00:00
AM
//
//    hu-HU culture:
//    12 --> Conversion Not Supported
//    17.2 --> Conversion Not Supported
//    False --> Conversion Not Supported
//    1/1/2010 12:00:00 AM --> 2010. 01. 01. 0:00:00
//    (CustomProvider retrieved.) today --> Bad Format
//    System.Collections.ArrayList --> Conversion Not Supported
//    c --> Conversion Not Supported
//    (CustomProvider retrieved.) 05/10/2009 6:13:18 PM --> 2009. 05. 10.
18:13:18
//    (CustomProvider retrieved.) September 8, 1899 --> 1899. 09. 08.
0:00:00
//
//    pt-PT culture:
//    12 --> Conversion Not Supported
//    17.2 --> Conversion Not Supported
//    False --> Conversion Not Supported
//    1/1/2010 12:00:00 AM --> 01-01-2010 0:00:00
//    (CustomProvider retrieved.) today --> Bad Format
//    System.Collections.ArrayList --> Conversion Not Supported
//    c --> Conversion Not Supported
//    (CustomProvider retrieved.) 05/10/2009 6:13:18 PM --> 05-10-2009
18:13:18
//    (CustomProvider retrieved.) September 8, 1899 --> 08-09-1899
0:00:00
```

Remarks

The return value is the result of invoking the IConvertible.ToDateTime method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a date, `provider` could supply culture-specific information about the notation used to represent that date. `provider` is involved in the conversion of `value` if the runtime type of `value` is a String, or if `value` is a user-defined type whose IConvertible.ToDateTime implementation makes use of `provider`. If the runtime type of `value` is String and `provider` is `null`, the CultureInfo object that represents the current culture is used.

See also

- Parsing Date and Time Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDateTime(SByte)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

```
C#

[System.CLSCompliant(false)]
public static DateTime ToDateTime (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(UInt32)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

```
C#

[System.CLSCompliant(false)]
public static DateTime ToDateTime (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

See also

- UInt32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Object)

Converts the value of the specified object to a DateTime object.

```
C#

public static DateTime ToDateTime (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

DateTime

The date and time equivalent of the value of `value`, or a date and time equivalent of DateTime.MinValue if `value` is `null`.

Exceptions

FormatException

`value` is not a valid date and time value.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

Examples

The following example calls the ToDateTime(Object) method with a variety of Object variables.

C#

```
using System;

public class ConversionToDateTime
{
   public static void Main()
   {
      // Try converting an integer.
      int number = 16352;
      ConvertToDateTime(number);

      // Convert a null.
      object obj = null;
      ConvertToDateTime(obj);

      // Convert a non-date string.
      string nonDateString = "monthly";
      ConvertToDateTime(nonDateString);

      // Try to convert various date strings.
      string dateString;
      dateString = "05/01/1996";
      ConvertToDateTime(dateString);
      dateString = "Tue Apr 28, 2009";
      ConvertToDateTime(dateString);
      dateString = "06 July 2008 7:32:47 AM";
      ConvertToDateTime(dateString);
      dateString = "17:32:47.003";
      ConvertToDateTime(dateString);
   }

   private static void ConvertToDateTime(object value)
   {
      DateTime convertedDate;
      try {
         convertedDate = Convert.ToDateTime(value);
         Console.WriteLine("'{0}' converts to {1}.", value,
convertedDate);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in the proper format.", value);
      }
      catch (InvalidCastException) {
         Console.WriteLine("Conversion of the {0} '{1}' is not
supported",
                           value.GetType().Name, value);
      }
   }
}
// The example displays the following output:
```

```
//       Conversion of the Int32 '16352' is not supported
//       '' converts to 1/1/0001 12:00:00 AM.
//       'monthly' is not in the proper format.
//       '05/01/1996' converts to 5/1/1996 12:00:00 AM.
//       'Tue Apr 28, 2009' converts to 4/28/2009 12:00:00 AM.
//       '06 July 2008 7:32:47 AM' converts to 7/6/2008 7:32:47 AM.
//       '17:32:47.003' converts to 5/28/2008 5:32:47 PM.
```

Remarks

For the conversion to succeed, the runtime type of the `value` parameter must be either a DateTime or a String, or `value` must be `null`. Otherwise, the method throws an InvalidCastException. In addition, if `value` is a string, it must contain a valid representation of a date and time value in the current culture or a FormatException is thrown.

The return value is the result of invoking the IConvertible.ToDateTime method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDateTime(Int16)

Calling this method always throws InvalidCastException.

C#

```
public static DateTime ToDateTime (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Int16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Int32)

Calling this method always throws InvalidCastException.

C#

```
public static DateTime ToDateTime (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Int64)

Calling this method always throws InvalidCastException.

C#

```
public static DateTime ToDateTime (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Double)

Calling this method always throws InvalidCastException.

C#

```
public static DateTime ToDateTime (double value);
```

Parameters

`value` Double

The double-precision floating-point value to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Double

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Decimal)

Calling this method always throws InvalidCastException.

C#

```
public static DateTime ToDateTime (decimal value);
```

Parameters

`value` Decimal

The number to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(DateTime)

Returns the specified DateTime object; no actual conversion is performed.

```
C#

public static DateTime ToDateTime (DateTime value);
```

Parameters

`value` DateTime

A date and time value.

Returns

DateTime

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Char)

Calling this method always throws InvalidCastException.

```
C#

public static DateTime ToDateTime (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Byte)

Calling this method always throws InvalidCastException.

```
C#
public static DateTime ToDateTime (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDateTime(Boolean)

Calling this method always throws InvalidCastException.

```
C#
public static DateTime ToDateTime (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

DateTime

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToDecimal Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a decimal number.

Overloads

Expand table

ToDecimal(Single)	Converts the value of the specified single-precision floating-point number to the equivalent decimal number.
ToDecimal(String)	Converts the specified string representation of a number to an equivalent decimal number.
ToDecimal(UInt16)	Converts the value of the specified 16-bit unsigned integer to an equivalent decimal number.
ToDecimal(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent decimal number, using the specified culture-specific formatting information.
ToDecimal(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent decimal number.
ToDecimal(Object, IFormatProvider)	Converts the value of the specified object to an equivalent decimal number, using the specified culture-specific formatting information.
ToDecimal(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent decimal number.
ToDecimal(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent decimal number.
ToDecimal(Object)	Converts the value of the specified object to an equivalent decimal number.
ToDecimal(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent decimal number.

ToDecimal(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent decimal number.
ToDecimal(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent decimal number.
ToDecimal(Double)	Converts the value of the specified double-precision floating-point number to an equivalent decimal number.
ToDecimal(Decimal)	Returns the specified decimal number; no actual conversion is performed.
ToDecimal(DateTime)	Calling this method always throws InvalidCastException.
ToDecimal(Char)	Calling this method always throws InvalidCastException.
ToDecimal(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent decimal number.
ToDecimal(Boolean)	Converts the specified Boolean value to the equivalent decimal number.

ToDecimal(Single)

Converts the value of the specified single-precision floating-point number to the equivalent decimal number.

C#

```
public static decimal ToDecimal (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

Examples

The following example tries to convert each element in an array of Single values to a Decimal value.

C#

```
float[] numbers = { Single.MinValue, -3e10f, -1093.54f, 0f, 1e-03f,
                    1034.23f, Single.MaxValue };
decimal result;

foreach (float number in numbers)
{
   try {
      result = Convert.ToDecimal(number);
      Console.WriteLine("Converted the Single value {0} to {1}.", number,
result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is out of range of the Decimal type.",
number);
   }
}
// The example displays the following output:
//       -3.402823E+38 is out of range of the Decimal type.
//       Converted the Single value -3E+10 to -30000000000.
//       Converted the Single value -1093.54 to -1093.54.
//       Converted the Single value 0 to 0.
//       Converted the Single value 0.001 to 0.001.
//       Converted the Single value 1034.23 to 1034.23.
//       3.402823E+38 is out of range of the Decimal type.
```

Remarks

The Decimal value returned by this method contains a maximum of seven significant digits. If the `value` parameter contains more than seven significant digits, it is rounded using rounding to nearest. The following example illustrates how the ToDecimal(Single) method uses rounding to nearest to return a Decimal value with seven significant digits.

C#

```
Console.WriteLine(Convert.ToDecimal(1234567500.12F));  // Displays
1234568000
Console.WriteLine(Convert.ToDecimal(1234568500.12F));  // Displays
1234568000
```

```
Console.WriteLine(Convert.ToDecimal(10.980365F));      // Displays
10.98036
Console.WriteLine(Convert.ToDecimal(10.980355F));      // Displays
10.98036
```

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(String)

Converts the specified string representation of a number to an equivalent decimal number.

C#

```
public static decimal ToDecimal (string? value);
```

Parameters

`value` String

A string that contains a number to convert.

Returns

Decimal

A decimal number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example illustrates the use of `ToDecimal`. It attempts to convert a String to a Decimal, and throws the possible exceptions that may arise during the conversion.

C#

```
public void ConvertStringDecimal(string stringVal) {
    decimal decimalVal = 0;

    try {
        decimalVal = System.Convert.ToDecimal(stringVal);
        System.Console.WriteLine(
            "The string as a decimal is {0}.", decimalVal);
    }
    catch (System.OverflowException){
        System.Console.WriteLine(
            "The conversion from string to decimal overflowed.");
    }
    catch (System.FormatException) {
        System.Console.WriteLine(
            "The string is not formatted as a decimal.");
    }
    catch (System.ArgumentNullException) {
        System.Console.WriteLine(
            "The string is null.");
    }

    // Decimal to string conversion will not overflow.
    stringVal = System.Convert.ToString(decimalVal);
    System.Console.WriteLine(
        "The decimal as a string is {0}.", stringVal);
}
```

Remarks

Using the ToDecimal(String) method is equivalent to passing `value` to the Decimal.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Decimal.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to an equivalent decimal number.

C#

```
[System.CLSCompliant(false)]
public static decimal ToDecimal (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Decimal

The decimal number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of 16-bit unsigned integers to Decimal values.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 12345, UInt16.MaxValue };
decimal result;

foreach (ushort number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the UInt16 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//       Converted the UInt16 value 0 to 0.
//       Converted the UInt16 value 121 to 121.
//       Converted the UInt16 value 12345 to 12345.
//       Converted the UInt16 value 65535 to 65535.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent decimal number, using the specified culture-specific formatting information.

```
C#

public static decimal ToDecimal (string? value, IFormatProvider?
provider);
```

Parameters

`value` String

A string that contains a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Decimal

A decimal number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example attempts to convert an array of strings to Decimal values by using NumberFormatInfo objects that represent two different cultures.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] values = { "123456789", "12345.6789", "12 345,6789",
                          "123,456.789", "123 456,789",
"123,456,789.0123",
                          "123 456 789,0123" };
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("fr-FR") };

      foreach (CultureInfo culture in cultures)
      {
         Console.WriteLine("String -> Decimal Conversion Using the {0}
Culture",
                           culture.Name);
         foreach (string value in values)
         {
            Console.Write("{0,20}  ->  ", value);
            try {
               Console.WriteLine(Convert.ToDecimal(value, culture));
            }
            catch (FormatException) {
               Console.WriteLine("FormatException");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       String -> Decimal Conversion Using the en-US Culture
//                  123456789  ->  123456789
//                 12345.6789  ->  12345.6789
//                12 345,6789  ->  FormatException
//                123,456.789  ->  123456.789
//                123 456,789  ->  FormatException
//           123,456,789.0123  ->  123456789.0123
//           123 456 789,0123  ->  FormatException
```

```
//
//       String -> Decimal Conversion Using the fr-FR Culture
//                   123456789  ->  123456789
//                  12345.6789  ->  FormatException
//                12 345,6789  ->  12345.6789
//                123,456.789  ->  FormatException
//                123 456,789  ->  123456.789
//            123,456,789.0123  ->  FormatException
//            123 456 789,0123  ->  123456789.0123
```

Remarks

The return value is the result of invoking the Decimal.Parse method on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Decimal.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(UInt64)

Important

This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent decimal number.

C#

```
[System.CLSCompliant(false)]
public static decimal ToDecimal (ulong value);
```

Parameters

value UInt64

The 64-bit unsigned integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of unsigned long integers to Decimal values.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 12345, UInt64.MaxValue };
decimal result;

foreach (ulong number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the UInt64 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//    Converted the UInt64 value 0 to 0.
//    Converted the UInt64 value 121 to 121.
//    Converted the UInt64 value 12345 to 12345.
```

```
//    Converted the UInt64 value 18446744073709551615 to
18446744073709551615.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Object, IFormatProvider)

Converts the value of the specified object to an equivalent decimal number, using the specified culture-specific formatting information.

C#

```
public static decimal ToDecimal (object? value, IFormatProvider?
provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Decimal

A decimal number that is equivalent to `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for a Decimal type.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example defines a `Temperature` class that implements the IConvertible interface.

C#

```
using System;
using System.Globalization;

public class Temperature : IConvertible
{
   private decimal m_Temp;

   public Temperature(decimal temperature)
   {
      this.m_Temp = temperature;
   }

   public decimal Celsius
   {
      get { return this.m_Temp; }
   }

   public decimal Kelvin
   {
      get { return this.m_Temp + 273.15m; }
```

```
    }

    public decimal Fahrenheit
    {
       get { return Math.Round((decimal) (this.m_Temp * 9 / 5 + 32), 2); }
    }

    public override string ToString()
    {
       return m_Temp.ToString("N2") + " °C";
    }

    // IConvertible implementations.
    public TypeCode GetTypeCode()
    {
       return TypeCode.Object;
    }

    public bool ToBoolean(IFormatProvider provider)
    {
       if (m_Temp == 0)
          return false;
       else
          return true;
    }

    public byte ToByte(IFormatProvider provider)
    {
       if (m_Temp < Byte.MinValue || m_Temp > Byte.MaxValue)
          throw new OverflowException(String.Format("{0} is out of range
of the Byte type.",
                                                     this.m_Temp));
       else
          return Decimal.ToByte(this.m_Temp);
    }

    public char ToChar(IFormatProvider provider)
    {
       throw new InvalidCastException("Temperature to Char conversion is
not supported.");
    }

    public DateTime ToDateTime(IFormatProvider provider)
    {
       throw new InvalidCastException("Temperature to DateTime conversion
is not supported.");
    }

    public decimal ToDecimal(IFormatProvider provider)
    {
       return this.m_Temp;
    }

    public double ToDouble(IFormatProvider provider)
    {
```

```
        return Decimal.ToDouble(this.m_Temp);
    }

    public short ToInt16(IFormatProvider provider)
    {
        if (this.m_Temp < Int16.MinValue || this.m_Temp > Int16.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Int16 type.",
                                                        this.m_Temp));
        else
            return Decimal.ToInt16(this.m_Temp);
    }

    public int ToInt32(IFormatProvider provider)
        {
        if (this.m_Temp < Int32.MinValue || this.m_Temp > Int32.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Int32 type.",
                                                        this.m_Temp));
        else
            return Decimal.ToInt32(this.m_Temp);
    }

    public long ToInt64(IFormatProvider provider)
    {
        if (this.m_Temp < Int64.MinValue || this.m_Temp > Int64.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the Int64 type.",
                                                        this.m_Temp));
        else
            return Decimal.ToInt64(this.m_Temp);
    }

    public sbyte ToSByte(IFormatProvider provider)
    {
        if (this.m_Temp < SByte.MinValue || this.m_Temp > SByte.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the SByte type.",
                                                        this.m_Temp));
        else
            return Decimal.ToSByte(this.m_Temp);
    }

    public float ToSingle(IFormatProvider provider)
    {
        return Decimal.ToSingle(this.m_Temp);
    }

    public string ToString(IFormatProvider provider)
    {
        return m_Temp.ToString("N2", provider) + " °C";
    }

    public object ToType(Type conversionType, IFormatProvider provider)
    {
```

```
      switch (Type.GetTypeCode(conversionType))
      {
         case TypeCode.Boolean:
            return this.ToBoolean(null);
         case TypeCode.Byte:
            return this.ToByte(null);
         case TypeCode.Char:
            return this.ToChar(null);
         case TypeCode.DateTime:
            return this.ToDateTime(null);
         case TypeCode.Decimal:
            return this.ToDecimal(null);
         case TypeCode.Double:
            return this.ToDouble(null);
         case TypeCode.Int16:
            return this.ToInt16(null);
         case TypeCode.Int32:
            return this.ToInt32(null);
         case TypeCode.Int64:
            return this.ToInt64(null);
         case TypeCode.Object:
            if (typeof(Temperature).Equals(conversionType))
               return this;
            else
               throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.",

conversionType.Name));
         case TypeCode.SByte:
            return this.ToSByte(null);
         case TypeCode.Single:
            return this.ToSingle(null);
         case TypeCode.String:
            return this.ToString(provider);
         case TypeCode.UInt16:
            return this.ToUInt16(null);
         case TypeCode.UInt32:
            return this.ToUInt32(null);
         case TypeCode.UInt64:
            return this.ToUInt64(null);
         default:
            throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
      }
   }

   public ushort ToUInt16(IFormatProvider provider)
   {
      if (this.m_Temp < UInt16.MinValue || this.m_Temp > UInt16.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt16 type.",
                                                     this.m_Temp));
      else
         return Decimal.ToUInt16(this.m_Temp);
   }
```

```
   public uint ToUInt32(IFormatProvider provider)
   {
      if (this.m_Temp < UInt32.MinValue || this.m_Temp > UInt32.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt32 type.",
                                                    this.m_Temp));
      else
         return Decimal.ToUInt32(this.m_Temp);
   }

   public ulong ToUInt64(IFormatProvider provider)
   {
      if (this.m_Temp < UInt64.MinValue || this.m_Temp > UInt64.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the UInt64 type.",
                                                    this.m_Temp));
      else
         return Decimal.ToUInt64(this.m_Temp);
   }
}
```

The following example shows that when a `Temperature` object is passed as a parameter to the ToDecimal(Object, IFormatProvider) method, the IConvertible.ToDecimal implementation of the `Temperature` class is called to perform the conversion.

C#

```
public class Example
{
   public static void Main()
   {
      Temperature cold = new Temperature(-40);
      Temperature freezing = new Temperature(0);
      Temperature boiling = new Temperature(100);

      Console.WriteLine(Convert.ToDecimal(cold, null));
      Console.WriteLine(Convert.ToDecimal(freezing, null));
      Console.WriteLine(Convert.ToDecimal(boiling, null));
   }
}
// The example dosplays the following output:
//       -40
//       0
//       100
```

Remarks

The return value is the result of invoking the IConvertible.ToDecimal method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent decimal number.

C#

```
[System.CLSCompliant(false)]
public static decimal ToDecimal (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of signed bytes to a Decimal value.

C#

```
sbyte[] numbers = { SByte.MinValue, -23, 0, 17, SByte.MaxValue };
decimal result;

foreach (sbyte number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the SByte value {0} to {1}.", number,
result);
}
//       Converted the SByte value -128 to -128.
//       Converted the SByte value -23 to -23.
//       Converted the SByte value 0 to 0.
//       Converted the SByte value 17 to 17.
//       Converted the SByte value 127 to 127.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent decimal number.

C#

```
[System.CLSCompliant(false)]
public static decimal ToDecimal (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts an array of unsigned integers to Decimal values.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 12345, UInt32.MaxValue };
decimal result;

foreach (uint number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the UInt32 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//       Converted the UInt32 value 0 to 0.
//       Converted the UInt32 value 121 to 121.
//       Converted the UInt32 value 12345 to 12345.
//       Converted the UInt32 value 4294967295 to 4294967295.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Object)

Converts the value of the specified object to an equivalent decimal number.

C#

```
public static decimal ToDecimal (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Decimal

A decimal number that is equivalent to `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for a Decimal type.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example tries to convert each element in an object array to a Decimal value.

C#

```
object[] values = { true, 'a', 123, 1.764e32, "9.78", "1e-02",
                    1.67e03, "A100", "1,033.67", DateTime.Now,
                    Double.MaxValue };
decimal result;

foreach (object value in values)
{
   try {
      result = Convert.ToDecimal(value);
      Console.WriteLine("Converted the {0} value {1} to {2}.",
                        value.GetType().Name, value, result);
   }
```

```
    catch (OverflowException) {
       Console.WriteLine("The {0} value {1} is out of range of the Decimal
type.",
                         value.GetType().Name, value);
    }
    catch (FormatException) {
       Console.WriteLine("The {0} value {1} is not recognized as a valid
Decimal value.",
                         value.GetType().Name, value);
    }
    catch (InvalidCastException) {
       Console.WriteLine("Conversion of the {0} value {1} to a Decimal is
not supported.",
                         value.GetType().Name, value);
    }
}
// The example displays the following output:
//    Converted the Boolean value True to 1.
//    Conversion of the Char value a to a Decimal is not supported.
//    Converted the Int32 value 123 to 123.
//    The Double value 1.764E+32 is out of range of the Decimal type.
//    Converted the String value 9.78 to 9.78.
//    The String value 1e-02 is not recognized as a valid Decimal value.
//    Converted the Double value 1670 to 1670.
//    The String value A100 is not recognized as a valid Decimal value.
//    Converted the String value 1,033.67 to 1033.67.
//    Conversion of the DateTime value 10/15/2008 05:40:42 PM to a
Decimal is not supported.
//    The Double value 1.79769313486232E+308 is out of range of the
Decimal type.
```

Remarks

The return value is the result of invoking the IConvertible.ToDecimal method of the underlying type of `value`.

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent decimal number.

C#

```
public static decimal ToDecimal (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Examples

The following example converts an Int64 value to a Decimal value.

C#

```
public void ConvertLongDecimal(long longVal) {

    decimal decimalVal;

    // Long to decimal conversion cannot overflow.
    decimalVal = System.Convert.ToDecimal(longVal);
    System.Console.WriteLine("{0} as a decimal is {1}",
            longVal, decimalVal);

    // Decimal to long conversion can overflow.
```

```
    try {
        longVal = System.Convert.ToInt64(decimalVal);
        System.Console.WriteLine("{0} as a long is {1}",
            decimalVal, longVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in decimal-to-long conversion.");
    }
}
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent decimal number.

C#

```
public static decimal ToDecimal (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Examples

The following example converts an array of signed integers to Decimal values.

C#

```
int[] numbers = { Int32.MinValue, -1000, 0, 1000, Int32.MaxValue };
decimal result;

foreach (int number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the Int32 value {0} to the Decimal value
{1}.",
                     number, result);
}
// The example displays the following output:
//    Converted the Int32 value -2147483648 to the Decimal value
-2147483648.
//    Converted the Int32 value -1000 to the Decimal value -1000.
//    Converted the Int32 value 0 to the Decimal value 0.
//    Converted the Int32 value 1000 to the Decimal value 1000.
//    Converted the Int32 value 2147483647 to the Decimal value
2147483647.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent decimal number.

C#

```
public static decimal ToDecimal (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Examples

The following example converts an array of 16-bit signed integers to Decimal values.

C#

```
short[] numbers = { Int16.MinValue, -1000, 0, 1000, Int16.MaxValue };
decimal result;

foreach (short number in numbers)
{
   result = Convert.ToDecimal(number);
   Console.WriteLine("Converted the Int16 value {0} to the Decimal value
{1}.",
                     number, result);
}
// The example displays the following output:
//       Converted the Int16 value -32768 to the Decimal value -32768.
//       Converted the Int16 value -1000 to the Decimal value -1000.
```

```
//       Converted the Int16 value 0 to the Decimal value 0.
//       Converted the Int16 value 1000 to the Decimal value 1000.
//       Converted the Int16 value 32767 to the Decimal value 32767.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Double)

Converts the value of the specified double-precision floating-point number to an equivalent decimal number.

C#

```
public static decimal ToDecimal (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Decimal

A decimal number that is equivalent to `value`.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

Examples

The following example converts a Double value to a Decimal value.

C#

```
public void ConvertDoubleDecimal(decimal decimalVal){

    double doubleVal;

    // Decimal to double conversion cannot overflow.
 doubleVal = System.Convert.ToDouble(decimalVal);
    System.Console.WriteLine("{0} as a double is: {1}",
            decimalVal, doubleVal);

    // Conversion from double to decimal can overflow.
    try
 {
       decimalVal = System.Convert.ToDecimal(doubleVal);
    System.Console.WriteLine ("{0} as a decimal is: {1}",
        doubleVal, decimalVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in double-to-double conversion.");
    }
}
```

Remarks

The Decimal value returned by this method contains a maximum of 15 significant digits. If the `value` parameter contains more than 15 significant digits, it is rounded using rounding to nearest. The following example illustrates how the Convert.ToDecimal(Double) method uses rounding to nearest to return a Decimal value with 15 significant digits.

C#

```
Console.WriteLine(Convert.ToDecimal(123456789012345500.12D));  //
Displays 123456789012346000
Console.WriteLine(Convert.ToDecimal(123456789012346500.12D));  //
Displays 123456789012346000
```

```
Console.WriteLine(Convert.ToDecimal(10030.12345678905D));      //
Displays 10030.123456789
Console.WriteLine(Convert.ToDecimal(10030.12345678915D));      //
Displays 10030.1234567892
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Decimal)

Returns the specified decimal number; no actual conversion is performed.

C#

```
public static decimal ToDecimal (decimal value);
```

Parameters

`value` Decimal

A decimal number.

Returns

Decimal

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static decimal ToDecimal (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Decimal

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDecimal(Char)

Calling this method always throws InvalidCastException.

C#

```
public static decimal ToDecimal (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Decimal

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Examples

The following example attempts to convert a Char value to Decimal, and throws InvalidCastException upon failure.

C#

```
public void ConvertCharDecimal(char charVal) {
    Decimal decimalVal = 0;

    // Char to decimal conversion is not supported and will always
    // throw an InvalidCastException.
    try {
        decimalVal = System.Convert.ToDecimal(charVal);
    }
    catch (System.InvalidCastException) {
        System.Console.WriteLine(
            "Char-to-Decimal conversion is not supported " +
            "by the .NET Framework.");
    }

    //Decimal to char conversion is also not supported.
    try {
        charVal = System.Convert.ToChar(decimalVal);
    }
    catch (System.InvalidCastException) {
        System.Console.WriteLine(
            "Decimal-to-Char conversion is not supported " +
            "by the .NET Framework.");
    }
}
```

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDecimal(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent decimal number.

C#

```
public static decimal ToDecimal (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Decimal

The decimal number that is equivalent to `value`.

Examples

The following example converts a Byte value to a Decimal value.

C#

```
public void ConvertByteDecimal(byte byteVal) {
    decimal decimalVal;

    // Byte to decimal conversion will not overflow.
    decimalVal = System.Convert.ToDecimal(byteVal);
    System.Console.WriteLine("The byte as a decimal is {0}.",
        decimalVal);

    // Decimal to byte conversion can overflow.
    try {
        byteVal = System.Convert.ToByte(decimalVal);
        System.Console.WriteLine("The Decimal as a byte is {0}.",
            byteVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "The decimal value is too large for a byte.");
    }
}
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDecimal(Boolean)

Converts the specified Boolean value to the equivalent decimal number.

C#

```
public static decimal ToDecimal (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Decimal

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example illustrates the conversion of Boolean to Decimal values.

C#

```
bool[] flags = { true, false };
decimal result;
```

```
foreach (bool flag in flags)
{
   result = Convert.ToDecimal(flag);
   Console.WriteLine("Converted {0} to {1}.", flag, result);
}
// The example displays the following output:
//       Converted True to 1.
//       Converted False to 0.
```

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToDouble Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a double-precision floating-point number.

Overloads

Expand table

ToDouble(Object, IFormatProvider)	Converts the value of the specified object to an double-precision floating-point number, using the specified culture-specific formatting information.
ToDouble(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent double-precision floating-point number.
ToDouble(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent double-precision floating-point number.
ToDouble(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent double-precision floating-point number.
ToDouble(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent double-precision floating-point number, using the specified culture-specific formatting information.
ToDouble(Single)	Converts the value of the specified single-precision floating-point number to an equivalent double-precision floating-point number.
ToDouble(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent double-precision floating-point number.
ToDouble(Object)	Converts the value of the specified object to a double-precision floating-point number.
ToDouble(String)	Converts the specified string representation of a number to an equivalent double-precision floating-point number.
ToDouble(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent double-precision floating-point number.

ToDouble(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent double-precision floating-point number.
ToDouble(Double)	Returns the specified double-precision floating-point number; no actual conversion is performed.
ToDouble(Decimal)	Converts the value of the specified decimal number to an equivalent double-precision floating-point number.
ToDouble(DateTime)	Calling this method always throws InvalidCastException.
ToDouble(Char)	Calling this method always throws InvalidCastException.
ToDouble(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent double-precision floating-point number.
ToDouble(Boolean)	Converts the specified Boolean value to the equivalent double-precision floating-point number.
ToDouble(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent double-precision floating-point number.

ToDouble(Object, IFormatProvider)

Converts the value of the specified object to an double-precision floating-point number, using the specified culture-specific formatting information.

C#

```
public static double ToDouble (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Double

A double-precision floating-point number that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for a Double type.

InvalidCastException

`value` does not implement the IConvertible interface.

OverflowException

`value` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the `ToDouble` method. This method returns an average of the array of Double values, using the object that implements IFormatProvider to determine how to calculate the average.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }
```

```
    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;

    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
```

```
        {
            if( index >= 0 && index < data.Count )
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;

        if( data.Count == 0 )
            return 0.0;

        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];

                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];
```

```
            default:
                return 0.0;
        }
    }

    // Get the AverageInfo object from the caller's format provider,
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }

    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
```

```
            GetAverageInfo( provider ).TypeOfAverage )
        {
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
            Byte.MinValue, Byte.MaxValue, provider ) );
    }

    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
```

```
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
    }

    public double ToDouble( IFormatProvider provider )
    {
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
```

```
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }

    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}

class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );

         // Display the dataset elements.
        if( ds.Count > 0 )
        {
            Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
            for( int iX = 1; iX < ds.Count; iX++ )
                Console.Write( ", {0}", ds[ iX ] );
            Console.WriteLine( "]\n" );
        }

        Console.WriteLine( fmt, "Convert.", "Default",
            "Geometric Mean", "Median");
        Console.WriteLine( fmt, "--------", "-------",
            "--------------", "------");
        Console.WriteLine( fmt, "ToBoolean",
            Convert.ToBoolean( ds, null ),
            Convert.ToBoolean( ds, geMean ),
            Convert.ToBoolean( ds, median ) );
        Console.WriteLine( fmt, "ToByte",
            Convert.ToByte( ds, null ),
            Convert.ToByte( ds, geMean ),
            Convert.ToByte( ds, median ) );
        Console.WriteLine( fmt, "ToChar",
```

```
            Convert.ToChar( ds, null ),
            Convert.ToChar( ds, geMean ),
            Convert.ToChar( ds, median ) );
        Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
            "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
            "ToDateTime", Convert.ToDateTime( ds, null ),
            Convert.ToDateTime( ds, geMean ),
            Convert.ToDateTime( ds, median ) );
        Console.WriteLine( fmt, "ToDecimal",
            Convert.ToDecimal( ds, null ),
            Convert.ToDecimal( ds, geMean ),
            Convert.ToDecimal( ds, median ) );
        Console.WriteLine( fmt, "ToDouble",
            Convert.ToDouble( ds, null ),
            Convert.ToDouble( ds, geMean ),
            Convert.ToDouble( ds, median ) );
        Console.WriteLine( fmt, "ToInt16",
            Convert.ToInt16( ds, null ),
            Convert.ToInt16( ds, geMean ),
            Convert.ToInt16( ds, median ) );
        Console.WriteLine( fmt, "ToInt32",
            Convert.ToInt32( ds, null ),
            Convert.ToInt32( ds, geMean ),
            Convert.ToInt32( ds, median ) );
        Console.WriteLine( fmt, "ToInt64",
            Convert.ToInt64( ds, null ),
            Convert.ToInt64( ds, geMean ),
            Convert.ToInt64( ds, median ) );
        Console.WriteLine( fmt, "ToSByte",
            Convert.ToSByte( ds, null ),
            Convert.ToSByte( ds, geMean ),
            Convert.ToSByte( ds, median ) );
        Console.WriteLine( fmt, "ToSingle",
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
```

```
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );

        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );

        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       103                  59                  67
ToChar                         g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                      103                  59                  67
ToInt32                      103                  59                  67
ToInt64                      103                  59                  67
ToSByte                      103                  59                  67
ToSingle                 103.4917            59.43321                67.3
ToUInt16                     103                  59                  67
ToUInt32                     103                  59                  67
ToUInt64                     103                  59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal               750999.89    631577.237188435            499999.5
ToDouble                750999.89    631577.237188435            499999.5
ToInt16                     32767               32767               32767
ToInt32                    751000              631577              500000
ToInt64                    751000              631577              500000
ToSByte                       127                 127                 127
ToSingle                 750999.9            631577.3            499999.5
ToUInt16                    65535               65535               65535
ToUInt32                   751000              631577              500000
```

```
ToUInt64                    751000              631577              500000
*/
```

Remarks

The return value is the result of invoking the IConvertible.ToDouble method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent double-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static double ToDouble (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of UInt64 values to a Double value.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 12345, UInt64.MaxValue };
double result;

foreach (ulong number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the UInt64 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//    Converted the UInt64 value 0 to 0.
//    Converted the UInt64 value 121 to 121.
//    Converted the UInt64 value 12345 to 12345.
//    Converted the UInt64 value 18446744073709551615 to
1.84467440737096E+19.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent double-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static double ToDouble (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of UInt32 values to a Double value.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 12345, UInt32.MaxValue };
double result;

foreach (uint number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the UInt32 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//       Converted the UInt32 value 0 to 0.
//       Converted the UInt32 value 121 to 121.
//       Converted the UInt32 value 12345 to 12345.
//       Converted the UInt32 value 4294967295 to 4294967295.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent double-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static double ToDouble (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of UInt16 values to a Double value.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 12345, UInt16.MaxValue };
double result;

foreach (ushort number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the UInt16 value {0} to {1}.",
                     number, result);
}
// The example displays the following output:
//       Converted the UInt16 value 0 to 0.
//       Converted the UInt16 value 121 to 121.
//       Converted the UInt16 value 12345 to 12345.
//       Converted the UInt16 value 65535 to 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent double-precision floating-point number, using the specified culture-specific formatting information.

```
C#
public static double ToDouble (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Double

A double-precision floating-point number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example converts string representations of Double values with the `ToDouble` method, using an IFormatProvider object.

C#

```
using System;
using System.Globalization;

class Example
{
    static void Main()
    {
        // Create a NumberFormatInfo object and set some of its
properties.
        NumberFormatInfo provider = new NumberFormatInfo();
        provider.NumberDecimalSeparator = ",";
        provider.NumberGroupSeparator = ".";
        provider.NumberGroupSizes = new int[] { 3 };

        // Define an array of numeric strings to convert.
        String[] values = { "123456789", "12345.6789", "12345,6789",
                             "123,456.789", "123.456,789",
                             "123,456,789.0123", "123.456.789,0123" };

        Console.WriteLine("Default Culture: {0}\n",
                          CultureInfo.CurrentCulture.Name);
        Console.WriteLine("{0,-22} {1,-20} {2,-20}\n", "String to
Convert",
                          "Default/Exception", "Provider/Exception");

        // Convert each string to a Double with and without the provider.
        foreach (var value in values) {
           Console.Write("{0,-22} ", value);
           try {
              Console.Write("{0,-20} ", Convert.ToDouble(value));
           }
           catch (FormatException e) {
              Console.Write("{0,-20} ", e.GetType().Name);
           }
           try {
              Console.WriteLine("{0,-20} ", Convert.ToDouble(value,
provider));
           }
           catch (FormatException e) {
              Console.WriteLine("{0,-20} ", e.GetType().Name);
           }
        }
```

```
    }
}
// The example displays the following output:
//       Default Culture: en-US
//
//       String to Convert      Default/Exception    Provider/Exception
//
//       123456789              123456789            123456789
//       12345.6789             12345.6789           123456789
//       12345,6789             123456789            12345.6789
//       123,456.789            123456.789           FormatException
//       123.456,789            FormatException      123456.789
//       123,456,789.0123       123456789.0123       FormatException
//       123.456.789,0123       FormatException      123456789.0123
```

Remarks

The return value is the result of invoking the Double.Parse method on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Double.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Single)

Converts the value of the specified single-precision floating-point number to an equivalent double-precision floating-point number.

```C#
public static double ToDouble (float value);
```

Parameters

`value` Single

The single-precision floating-point number.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Examples

The following example converts a Single value to a Double value.

```C#
public void CovertDoubleFloat(double doubleVal) {
    float floatVal = 0;

    // Double to float conversion cannot overflow.
        floatVal = System.Convert.ToSingle(doubleVal);
        System.Console.WriteLine("{0} as a float is {1}",
            doubleVal, floatVal);

    // Conversion from float to double cannot overflow.
    doubleVal = System.Convert.ToDouble(floatVal);
    System.Console.WriteLine("{0} as a double is: {1}",
        floatVal, doubleVal);
}
```

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent double-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static double ToDouble (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Double

The 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of SByte values to a Double value.

C#

```
sbyte[] numbers = { SByte.MinValue, -23, 0, 17, SByte.MaxValue };
double result;

foreach (sbyte number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the SByte value {0} to {1}.", number,
result);
}
//       Converted the SByte value -128 to -128.
//       Converted the SByte value -23 to -23.
//       Converted the SByte value 0 to 0.
//       Converted the SByte value 17 to 17.
//       Converted the SByte value 127 to 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Object)

Converts the value of the specified object to a double-precision floating-point number.

C#

```
public static double ToDouble (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Double

A double-precision floating-point number that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for a Double type.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example attempts to convert each value in an object array to a Double.

C#

```
object[] values = { true, 'a', 123, 1.764e32f, "9.78", "1e-02",
                    1.67e03f, "A100", "1,033.67", DateTime.Now,
                    Decimal.MaxValue };
double result;

foreach (object value in values)
{
   try {
      result = Convert.ToDouble(value);
      Console.WriteLine("Converted the {0} value {1} to {2}.",
                        value.GetType().Name, value, result);
   }
```

```
    catch (FormatException) {
       Console.WriteLine("The {0} value {1} is not recognized as a valid
Double value.",
                         value.GetType().Name, value);
    }
    catch (InvalidCastException) {
       Console.WriteLine("Conversion of the {0} value {1} to a Double is
not supported.",
                         value.GetType().Name, value);
    }
}
// The example displays the following output:
//    Converted the Boolean value True to 1.
//    Conversion of the Char value a to a Double is not supported.
//    Converted the Int32 value 123 to 123.
//    Converted the Single value 1.764E+32 to 1.76399995098587E+32.
//    Converted the String value 9.78 to 9.78.
//    Converted the String value 1e-02 to 0.01.
//    Converted the Single value 1670 to 1670.
//    The String value A100 is not recognized as a valid Double value.
//    Converted the String value 1,033.67 to 1033.67.
//    Conversion of the DateTime value 10/21/2008 07:12:12 AM to a Double
is not supported.
//    Converted the Decimal value 79228162514264337593543950335 to
7.92281625142643E+28.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToDouble implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(String)

Converts the specified string representation of a number to an equivalent double-precision floating-point number.

```
C#

public static double ToDouble (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

Double

A double-precision floating-point number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example attempts to convert each element in an array of numeric strings to a Double. The example's output is from a system whose current culture is en-US.

```
C#

using System;

public class Example
{
   public static void Main()
```

```
    {
       string[] values= { "-1,035.77219", "1AFF", "1e-35",
                          "1,635,592,999,999,999,999,999,999", "-17.455",
                          "190.34001", "1.29e325"};
       double result;

       foreach (string value in values)
       {
          try {
             result = Convert.ToDouble(value);
             Console.WriteLine("Converted '{0}' to {1}.", value, result);
          }
          catch (FormatException) {
             Console.WriteLine("Unable to convert '{0}' to a Double.",
value);
          }
          catch (OverflowException) {
             Console.WriteLine("'{0}' is outside the range of a Double.",
value);
          }
       }
    }
}
// The example displays the following output:
//       Converted '-1,035.77219' to -1035.77219.
//       Unable to convert '1AFF' to a Double.
//       Converted '1e-35' to 1E-35.
//       Converted '1,635,592,999,999,999,999,999,999' to 1.635593E+24.
//       Converted '-17.455' to -17.455.
//       Converted '190.34001' to 190.34001.
//       '1.29e325' is outside the range of a Double.
```

Remarks

Using the ToDouble(String) method is equivalent to passing `value` to the Double.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Double.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent double-precision floating-point number.

C#

```
public static double ToDouble (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Examples

The following example converts an Int32 value to a Double value.

C#

```
public void ConvertDoubleInt(double doubleVal) {

    int     intVal = 0;
    // Double to int conversion can overflow.
    try {
        intVal = System.Convert.ToInt32(doubleVal);
        System.Console.WriteLine("{0} as an int is: {1}",
```

```
            doubleVal, intVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in double-to-int conversion.");
    }

    // Int to double conversion cannot overflow.
    doubleVal = System.Convert.ToDouble(intVal);
    System.Console.WriteLine("{0} as a double is: {1}",
        intVal, doubleVal);
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent double-precision floating-point number.

C#

```
public static double ToDouble (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Double

A double-precision floating-point number equivalent to `value`.

Examples

The following example converts each element in an array of 16-bit signed integers to a Double value.

C#

```
short[] numbers = { Int16.MinValue, -1032, 0, 192, Int16.MaxValue };
double result;

foreach (short number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the UInt16 value {0} to {1}.",
                     number, result);
}
//       Converted the UInt16 value -32768 to -32768.
//       Converted the UInt16 value -1032 to -1032.
//       Converted the UInt16 value 0 to 0.
//       Converted the UInt16 value 192 to 192.
//       Converted the UInt16 value 32767 to 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Double)

Returns the specified double-precision floating-point number; no actual conversion is performed.

```
C#
```

```
public static double ToDouble (double value);
```

Parameters

`value` Double

The double-precision floating-point number to return.

Returns

Double

`value` is returned unchanged.

See also

- Double

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Decimal)

Converts the value of the specified decimal number to an equivalent double-precision floating-point number.

```
C#
```

```
public static double ToDouble (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Examples

The following example converts a Decimal value to a Double value.

C#

```
public void ConvertDoubleDecimal(decimal decimalVal){

    double doubleVal;

    // Decimal to double conversion cannot overflow.
 doubleVal = System.Convert.ToDouble(decimalVal);
    System.Console.WriteLine("{0} as a double is: {1}",
            decimalVal, doubleVal);

    // Conversion from double to decimal can overflow.
    try
 {
       decimalVal = System.Convert.ToDecimal(doubleVal);
    System.Console.WriteLine ("{0} as a decimal is: {1}",
        doubleVal, decimalVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in double-to-double conversion.");
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static double ToDouble (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Double

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDouble(Char)

Calling this method always throws InvalidCastException.

C#

```
public static double ToDouble (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Double

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToDouble(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent double-precision floating-point number.

C#

```
public static double ToDouble (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Double

The double-precision floating-point number that is equivalent to `value`.

Examples

The following example converts a Byte value to a Double value.

C#

```
public void ConvertDoubleByte(double doubleVal) {
    byte    byteVal = 0;

    // Double to byte conversion can overflow.
    try {
        byteVal = System.Convert.ToByte(doubleVal);
        System.Console.WriteLine("{0} as a byte is: {1}.",
            doubleVal, byteVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Overflow in double-to-byte conversion.");
```

```
    }

    // Byte to double conversion cannot overflow.
    doubleVal = System.Convert.ToDouble(byteVal);
    System.Console.WriteLine("{0} as a double is: {1}.",
        byteVal, doubleVal);
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Boolean)

Converts the specified Boolean value to the equivalent double-precision floating-point number.

C#

```
public static double ToDouble (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Double

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example converts a Boolean value to a Double value.

C#

```
public void ConvertDoubleBool(double doubleVal) {
    bool    boolVal;
    // Double to bool conversion cannot overflow.
    boolVal = System.Convert.ToBoolean(doubleVal);
    System.Console.WriteLine("{0} as a Boolean is: {1}.",
        doubleVal, boolVal);

    // bool to double conversion cannot overflow.
    doubleVal = System.Convert.ToDouble(boolVal);
    System.Console.WriteLine("{0} as a double is: {1}.",
        boolVal, doubleVal);
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToDouble(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent double-precision floating-point number.

C#

```
public static double ToDouble (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Double

A double-precision floating-point number that is equivalent to `value`.

Examples

The following example converts each element in an array of signed long integers to a Double value.

C#

```
long[] numbers = { Int64.MinValue, -903, 0, 172, Int64.MaxValue};
double result;

foreach (long number in numbers)
{
   result = Convert.ToDouble(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int64 value '-9223372036854775808' to the Double
value -9.22337203685478E+18.
//    Converted the Int64 value '-903' to the Double value -903.
//    Converted the Int64 value '0' to the Double value 0.
//    Converted the Int64 value '172' to the Double value 172.
//    Converted the Int64 value '9223372036854775807' to the Double value
9.22337203685478E+18.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToHexString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToHexString(ReadOnlySpan<Byte>)	Converts a span of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.
ToHexString(Byte[], Int32, Int32)	Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters. Parameters specify the subset as an offset in the input array and the number of elements in the array to convert.
ToHexString(Byte[])	Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.

ToHexString(ReadOnlySpan<Byte>)

Converts a span of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.

C#

```
public static string ToHexString (ReadOnlySpan<byte> bytes);
```

Parameters

`bytes` ReadOnlySpan<Byte>

A span of 8-bit unsigned integers.

Returns

String

The string representation in hex of the elements in `bytes`.

Exceptions

ArgumentOutOfRangeException

`bytes` is too large to be encoded.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToHexString(Byte[], Int32, Int32)

Converts a subset of an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters. Parameters specify the subset as an offset in the input array and the number of elements in the array to convert.

```
C#

public static string ToHexString (byte[] inArray, int offset, int
length);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

`offset` Int32

An offset in `inArray`.

`length` Int32

The number of elements of `inArray` to convert.

Returns

String

The string representation in hex of `length` elements of `inArray`, starting at position `offset`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentOutOfRangeException

`offset` or `length` is negative.

ArgumentOutOfRangeException

`offset` plus `length` is greater than the length of `inArray`.

ArgumentOutOfRangeException

`inArray` is too large to be encoded.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToHexString(Byte[])

Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with uppercase hex characters.

C#

```
public static string ToHexString (byte[] inArray);
```

Parameters

`inArray` Byte[]

An array of 8-bit unsigned integers.

Returns

String

The string representation in hex of the elements in `inArray`.

Exceptions

ArgumentNullException

`inArray` is `null`.

ArgumentOutOfRangeException

`inArray` is too large to be encoded.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToInt16 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 16-bit signed integer.

Overloads

Expand table

ToInt16(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 16-bit signed integer.
ToInt16(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 16-bit signed integer, using the specified culture-specific formatting information.
ToInt16(Object, IFormatProvider)	Converts the value of the specified object to a 16-bit signed integer, using the specified culture-specific formatting information.
ToInt16(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit signed integer.
ToInt16(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit signed integer.
ToInt16(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 16-bit signed integer.
ToInt16(String)	Converts the specified string representation of a number to an equivalent 16-bit signed integer.
ToInt16(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 16-bit signed integer.
ToInt16(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 16-bit signed integer.
ToInt16(Int16)	Returns the specified 16-bit signed integer; no actual conversion is performed.

ToInt16(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 16-bit signed integer.
ToInt16(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 16-bit signed integer.
ToInt16(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 16-bit signed integer.
ToInt16(Decimal)	Converts the value of the specified decimal number to an equivalent 16-bit signed integer.
ToInt16(DateTime)	Calling this method always throws InvalidCastException.
ToInt16(Char)	Converts the value of the specified Unicode character to the equivalent 16-bit signed integer.
ToInt16(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit signed integer.
ToInt16(Boolean)	Converts the specified Boolean value to the equivalent 16-bit signed integer.
ToInt16(Object)	Converts the value of the specified object to a 16-bit signed integer.

ToInt16(Single)

Converts the value of the specified single-precision floating-point number to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Int16

`value`, rounded to the nearest 16-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue or less than Int16.MinValue.

Examples

The following example attempts to convert each element in an array of Single values to a 16-bit signed integer.

C#

```
float[] values = { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                   0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
short result;

foreach (float value in values)
{
   try {
      result = Convert.ToInt16(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int16 type.",
value);
   }
}
// The example displays the following output:
//    -3.4028235E+38 is outside the range of the Int16 type.
//    -1.38E+10 is outside the range of the Int16 type.
//    Converted the Single value -1023.299 to the Int16 value -1023.
//    Converted the Single value -12.98 to the Int16 value -13.
//    Converted the Single value 0 to the Int16 value 0.
//    Converted the Single value 9.113E-16 to the Int16 value 0.
//    Converted the Single value 103.919 to the Int16 value 104.
//    Converted the Single value 17834.191 to the Int16 value 17834.
//    3.4028235E+38 is outside the range of the Int16 type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent 16-bit signed integer, using the specified culture-specific formatting information.

C#

```
public static short ToInt16 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int16

A 16-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example converts string representations of 16-bit integers with the `ToInt16` method, using an IFormatProvider object.

C#

```
// Example of the Convert.ToInt16( string ) and
// Convert.ToInt16( string, IFormatProvider ) methods.
using System;
using System.Globalization;

class ToInt16ProviderDemo
{
    static string format = "{0,-20}{1,-20}{2}";

     // Get the exception type name; remove the namespace prefix.
    static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    static void ConvertToInt16( string numericStr,
        IFormatProvider provider )
    {
        object defaultValue;
        object providerValue;

        // Convert numericStr to Int16 without a format provider.
        try
        {
            defaultValue = Convert.ToInt16( numericStr );
        }
        catch( Exception ex )
        {
            defaultValue = GetExceptionType( ex );
        }

        // Convert numericStr to Int16 with a format provider.
        try
        {
            providerValue = Convert.ToInt16( numericStr, provider );
        }
        catch( Exception ex )
        {
```

```
            providerValue = GetExceptionType( ex );
        }

        Console.WriteLine( format, numericStr,
            defaultValue, providerValue );
    }

    public static void Main( )
    {
        // Create a NumberFormatInfo object and set several of its
        // properties that apply to numbers.
        NumberFormatInfo provider = new NumberFormatInfo();

        // These properties affect the conversion.
        provider.NegativeSign = "neg ";
        provider.PositiveSign = "pos ";

        // These properties do not affect the conversion.
        // The input string cannot have decimal and group separators.
        provider.NumberDecimalSeparator = ".";
        provider.NumberGroupSeparator = ",";
        provider.NumberGroupSizes = new int[ ] { 3 };
        provider.NumberNegativePattern = 0;

        Console.WriteLine("This example of\n" +
            "  Convert.ToInt16( string ) and \n" +
            "  Convert.ToInt16( string, IFormatProvider ) " +
            "\ngenerates the following output. It converts " +
            "several strings to \nshort values, using " +
            "default formatting or a NumberFormatInfo object.\n" );
        Console.WriteLine( format, "String to convert",
            "Default/exception", "Provider/exception" );
        Console.WriteLine( format, "-----------------",
            "-----------------", "-----------------" );

        // Convert strings, with and without an IFormatProvider.
        ConvertToInt16( "12345", provider );
        ConvertToInt16( "+12345", provider );
        ConvertToInt16( "pos 12345", provider );
        ConvertToInt16( "-12345", provider );
        ConvertToInt16( "neg 12345", provider );
        ConvertToInt16( "12345.", provider );
        ConvertToInt16( "12,345", provider );
        ConvertToInt16( "(12345)", provider );
        ConvertToInt16( "32768", provider );
        ConvertToInt16( "-32769", provider );
    }
}

/*
This example of
  Convert.ToInt16( string ) and
  Convert.ToInt16( string, IFormatProvider )
generates the following output. It converts several strings to
short values, using default formatting or a NumberFormatInfo object.
```

```
String to convert   Default/exception   Provider/exception
-----------------   -----------------   ------------------
12345               12345               12345
+12345              12345               FormatException
pos 12345           FormatException     12345
-12345              -12345              FormatException
neg 12345           FormatException     -12345
12345.              FormatException     FormatException
12,345              FormatException     FormatException
(12345)             FormatException     FormatException
32768               OverflowException   OverflowException
-32769              OverflowException   FormatException
*/
```

Remarks

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Int16.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Object, IFormatProvider)

Converts the value of the specified object to a 16-bit signed integer, using the specified culture-specific formatting information.

```
C#

public static short ToInt16 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for an Int16 type.

InvalidCastException

`value` does not implement IConvertible.

OverflowException

`value` represents a number that is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the `ToInt16` method.

This method returns an average of the array of Double values, using the object that implements IFormatProvider to determine how to calculate the average.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }

    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
```

```
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;

    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
        {
            if( index >= 0 && index < data.Count )
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;
```

```
        if( data.Count == 0 )
            return 0.0;

        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];

                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];

            default:
                return 0.0;
        }
    }

    // Get the AverageInfo object from the caller's format provider,
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
```

```
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }

    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
            GetAverageInfo( provider ).TypeOfAverage )
        {
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
```

```
            Byte.MinValue, Byte.MaxValue, provider ) );
    }

    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
    }

    public double ToDouble( IFormatProvider provider )
    {
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
```

```
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }

    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}
```

```
class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );

         // Display the dataset elements.
        if( ds.Count > 0 )
        {
            Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
            for( int iX = 1; iX < ds.Count; iX++ )
                Console.Write( ", {0}", ds[ iX ] );
            Console.WriteLine( "]\n" );
        }

        Console.WriteLine( fmt, "Convert.", "Default",
            "Geometric Mean", "Median");
        Console.WriteLine( fmt, "--------", "-------",
            "--------------", "------");
        Console.WriteLine( fmt, "ToBoolean",
            Convert.ToBoolean( ds, null ),
            Convert.ToBoolean( ds, geMean ),
            Convert.ToBoolean( ds, median ) );
        Console.WriteLine( fmt, "ToByte",
            Convert.ToByte( ds, null ),
            Convert.ToByte( ds, geMean ),
            Convert.ToByte( ds, median ) );
        Console.WriteLine( fmt, "ToChar",
            Convert.ToChar( ds, null ),
            Convert.ToChar( ds, geMean ),
            Convert.ToChar( ds, median ) );
        Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
            "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
            "ToDateTime", Convert.ToDateTime( ds, null ),
            Convert.ToDateTime( ds, geMean ),
            Convert.ToDateTime( ds, median ) );
        Console.WriteLine( fmt, "ToDecimal",
            Convert.ToDecimal( ds, null ),
            Convert.ToDecimal( ds, geMean ),
            Convert.ToDecimal( ds, median ) );
        Console.WriteLine( fmt, "ToDouble",
            Convert.ToDouble( ds, null ),
            Convert.ToDouble( ds, geMean ),
            Convert.ToDouble( ds, median ) );
        Console.WriteLine( fmt, "ToInt16",
            Convert.ToInt16( ds, null ),
            Convert.ToInt16( ds, geMean ),
            Convert.ToInt16( ds, median ) );
        Console.WriteLine( fmt, "ToInt32",
            Convert.ToInt32( ds, null ),
            Convert.ToInt32( ds, geMean ),
```

```
            Convert.ToInt32( ds, median ) );
        Console.WriteLine( fmt, "ToInt64",
            Convert.ToInt64( ds, null ),
            Convert.ToInt64( ds, geMean ),
            Convert.ToInt64( ds, median ) );
        Console.WriteLine( fmt, "ToSByte",
            Convert.ToSByte( ds, null ),
            Convert.ToSByte( ds, geMean ),
            Convert.ToSByte( ds, median ) );
        Console.WriteLine( fmt, "ToSingle",
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );

        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );

        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
```

```
ToByte                      103                  59                  67
ToChar                        g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                     103                  59                  67
ToInt32                     103                  59                  67
ToInt64                     103                  59                  67
ToSByte                     103                  59                  67
ToSingle                103.4917            59.43321                67.3
ToUInt16                    103                  59                  67
ToUInt32                    103                  59                  67
ToUInt64                    103                  59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal              750999.89    631577.237188435            499999.5
ToDouble               750999.89    631577.237188435            499999.5
ToInt16                    32767               32767               32767
ToInt32                   751000              631577              500000
ToInt64                   751000              631577              500000
ToSByte                      127                 127                 127
ToSingle                750999.9            631577.3            499999.5
ToUInt16                   65535               65535               65535
ToUInt32                  751000              631577              500000
ToUInt64                  751000              631577              500000
*/
```

Remarks

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static short ToInt16 (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int16.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned long integers to a 16-bit signed integer.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
short result;

foreach (ulong number in numbers)
{
   try {
      result = Convert.ToInt16(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt64 value 0 to a Int16 value 0.
//    Converted the UInt64 value 121 to a Int16 value 121.
//    Converted the UInt64 value 340 to a Int16 value 340.
//    The UInt64 value 18446744073709551615 is outside the range of the
Int16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToInt16(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit signed integer.

```
C#

[System.CLSCompliant(false)]
public static short ToInt16 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int16.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned integers to a 16-bit signed integer.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
short result;

foreach (uint number in numbers)
{
   try {
      result = Convert.ToInt16(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt32 value 0 to a Int16 value 0.
//    Converted the UInt32 value 121 to a Int16 value 121.
//    Converted the UInt32 value 340 to a Int16 value 340.
//    The UInt32 value 4294967295 is outside the range of the Int16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 16-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static short ToInt16 (ushort value);
```

Parameters

value UInt16

The 16-bit unsigned integer to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int16.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned 16-bit integers to a 16-bit signed integer.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
short result;
foreach (ushort number in numbers)
{
   try {
      result = Convert.ToInt16(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
```

```
        Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                          number.GetType().Name, number);
    }
}
// The example displays the following output:
//       Converted the UInt16 value 0 to a Int16 value 0.
//       Converted the UInt16 value 121 to a Int16 value 121.
//       Converted the UInt16 value 340 to a Int16 value 340.
//       The UInt16 value 65535 is outside the range of the Int16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(String)

Converts the specified string representation of a number to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example converts string representations of 16-bit integers with the `ToInt16` method, using default formatting.

C#

```
// Example of the Convert.ToInt16( string ) and
// Convert.ToInt16( string, IFormatProvider ) methods.
using System;
using System.Globalization;

class ToInt16ProviderDemo
{
    static string format = "{0,-20}{1,-20}{2}";

     // Get the exception type name; remove the namespace prefix.
    static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    static void ConvertToInt16( string numericStr,
        IFormatProvider provider )
    {
        object defaultValue;
        object providerValue;

        // Convert numericStr to Int16 without a format provider.
        try
        {
            defaultValue = Convert.ToInt16( numericStr );
        }
```

```
        catch( Exception ex )
        {
            defaultValue = GetExceptionType( ex );
        }

        // Convert numericStr to Int16 with a format provider.
        try
        {
            providerValue = Convert.ToInt16( numericStr, provider );
        }
        catch( Exception ex )
        {
            providerValue = GetExceptionType( ex );
        }

        Console.WriteLine( format, numericStr,
            defaultValue, providerValue );
    }

    public static void Main( )
    {
        // Create a NumberFormatInfo object and set several of its
        // properties that apply to numbers.
        NumberFormatInfo provider = new NumberFormatInfo();

        // These properties affect the conversion.
        provider.NegativeSign = "neg ";
        provider.PositiveSign = "pos ";

        // These properties do not affect the conversion.
        // The input string cannot have decimal and group separators.
        provider.NumberDecimalSeparator = ".";
        provider.NumberGroupSeparator = ",";
        provider.NumberGroupSizes = new int[ ] { 3 };
        provider.NumberNegativePattern = 0;

        Console.WriteLine("This example of\n" +
            "  Convert.ToInt16( string ) and \n" +
            "  Convert.ToInt16( string, IFormatProvider ) " +
            "\ngenerates the following output. It converts " +
            "several strings to \nshort values, using " +
            "default formatting or a NumberFormatInfo object.\n" );
        Console.WriteLine( format, "String to convert",
            "Default/exception", "Provider/exception" );
        Console.WriteLine( format, "-----------------",
            "-----------------", "------------------" );

        // Convert strings, with and without an IFormatProvider.
        ConvertToInt16( "12345", provider );
        ConvertToInt16( "+12345", provider );
        ConvertToInt16( "pos 12345", provider );
        ConvertToInt16( "-12345", provider );
        ConvertToInt16( "neg 12345", provider );
        ConvertToInt16( "12345.", provider );
        ConvertToInt16( "12,345", provider );
```

```
        ConvertToInt16( "(12345)", provider );
        ConvertToInt16( "32768", provider );
        ConvertToInt16( "-32769", provider );
    }
}

/*
This example of
  Convert.ToInt16( string ) and
  Convert.ToInt16( string, IFormatProvider )
generates the following output. It converts several strings to
short values, using default formatting or a NumberFormatInfo object.

String to convert   Default/exception   Provider/exception
-----------------   -----------------   ------------------
12345               12345               12345
+12345              12345               FormatException
pos 12345           FormatException     12345
-12345              -12345              FormatException
neg 12345           FormatException     -12345
12345.              FormatException     FormatException
12,345              FormatException     FormatException
(12345)             FormatException     FormatException
32768               OverflowException   OverflowException
-32769              OverflowException   FormatException
*/
```

Remarks

Using the ToInt16(String) method is equivalent to passing `value` to the Int16.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Int16.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(String, Int32)

Converts the string representation of a number in a specified base to an equivalent 16-bit signed integer.

```
C#

public static short ToInt16 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

Int16

A 16-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

ArgumentOutOfRangeException

`value` is Empty.

FormatException

`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example attempts to interpret each element in a string array as a hexadecimal string and to convert it to a 16-bit signed integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] hexStrings = { "8000", "0FFF", "f000", "00A30", "D",
"-13",
                              "9AC61", "GAD" };
      foreach (string hexString in hexStrings)
      {
         try {
            short number = Convert.ToInt16(hexString, 16);
            Console.WriteLine("Converted '{0}' to {1:N0}.", hexString,
number);
         }
         catch (FormatException) {
            Console.WriteLine("'{0}' is not in the correct format for a
```

```
hexadecimal number.",
                                hexString);
         }
         catch (OverflowException) {
            Console.WriteLine("'{0}' is outside the range of an Int16.",
hexString);
         }
         catch (ArgumentException) {
            Console.WriteLine("'{0}' is invalid in base 16.", hexString);
         }
      }
   }
}
// The example displays the following output:
//       Converted '8000' to -32,768.
//       Converted '0FFF' to 4,095.
//       Converted 'f000' to -4,096.
//       Converted '00A30' to 2,608.
//       Converted 'D' to 13.
//       '-13' is invalid in base 16.
//       '9AC61' is outside the range of an Int16.
//       'GAD' is not in the correct format for a hexadecimal number.
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the negative sign is not supported for non-base 10 numeric representations, the ToInt16(String, Int32) method assumes that negative numbers use two's complement representation. In other words, the method always interprets the highest-order binary bit of an integer (bit 15) as its sign bit. As a result, it is possible to write code in which a non-base 10 number that is out of the range of the Int16 data type is converted to an Int16 value without the method throwing an exception. The following example increments Int16.MaxValue by one, converts the resulting number to its hexadecimal string representation, and then calls the ToInt16(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x8000 converts to -32768."

C#

```
// Create a hexadecimal value out of range of the Int16 type.
string value = Convert.ToString((int) short.MaxValue + 1, 16);
// Convert it back to a number.
try
{
   short number = Convert.ToInt16(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
```

```
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to a 16-bit integer.",
value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method is using the appropriate numeric representation to interpret a particular value. As the following example illustrates, you can ensure that the method handles overflows appropriately by first retrieving the sign of the numeric value before converting it to its hexadecimal string representation. Throw an exception if the original value was positive but the conversion back to an integer yields a negative value.

C#

```
// Create a hexadecimal value out of range of the Short type.
int sourceNumber = (int) short.MaxValue + 1;
bool isNegative = (Math.Sign(sourceNumber) == -1);
string value = Convert.ToString(sourceNumber, 16);
short targetNumber;
try
{
   targetNumber = Convert.ToInt16(value, 16);
   if (! isNegative && ((targetNumber & 0x8000) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to a 16-bit integer.",
value);
}
// Displays the following to the console:
//    Unable to convert '0x8000' to a 16-bit integer.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 16-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static short ToInt16 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Int16

A 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of signed bytes to a 16-bit signed integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
short result;

foreach (sbyte number in numbers)
{
   result = Convert.ToInt16(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the SByte value -128 to the Int16 value -128.
//       Converted the SByte value -1 to the Int16 value -1.
//       Converted the SByte value 0 to the Int16 value 0.
//       Converted the SByte value 10 to the Int16 value 10.
//       Converted the SByte value 127 to the Int16 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Int16)

Returns the specified 16-bit signed integer; no actual conversion is performed.

C#

```
public static short ToInt16 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to return.

Returns

Int16

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue or less than Int16.MinValue.

Examples

The following example attempts to convert each element in an array of long integers to a 16-bit signed integer.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
short result;

foreach (long number in numbers)
{
   try {
      result = Convert.ToInt16(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
Int16 type.
//    Converted the Int64 value -1 to the Int16 value -1.
//    Converted the Int64 value 0 to the Int16 value 0.
//    Converted the Int64 value 121 to the Int16 value 121.
//    Converted the Int64 value 340 to the Int16 value 340.
//    The Int64 value 9223372036854775807 is outside the range of the
Int16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Int16

The 16-bit signed integer equivalent of `value`.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue or less than Int16.MinValue.

Examples

The following example attempts to convert each element in an array of integers to a 16-bit signed integer.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
short result;
```

```
foreach (int number in numbers)
{
   try {
      result = Convert.ToInt16(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int32 value -2147483648 is outside the range of the Int16 type.
//    Converted the Int32 value -1 to a Int16 value -1.
//    Converted the Int32 value 0 to a Int16 value 0.
//    Converted the Int32 value 121 to a Int16 value 121.
//    Converted the Int32 value 340 to a Int16 value 340.
//    The Int32 value 2147483647 is outside the range of the Int16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Double)

Converts the value of the specified double-precision floating-point number to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Int16

`value`, rounded to the nearest 16-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue or less than Int16.MinValue.

Examples

The following example converts each element in an array of Double values to a 16-bit signed integer. The example illustrates that any fractional part of a Double value is rounded before performing the conversion.

C#

```
double[] values = { Double.MinValue, -1.38e10, -1023.299, -12.98,
                    0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
short result;

foreach (double value in values)
{
   try {
      result = Convert.ToInt16(value);
      Console.WriteLine("Converted {0} to {1}.", value, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("{0} is outside the range of the Int16 type.",
value);
   }
}
//       -1.79769313486232E+308 is outside the range of the Int16 type.
```

```
//       -13800000000 is outside the range of the Int16 type.
//       Converted -1023.299 to -1023.
//       Converted -12.98 to -13.
//       Converted 0 to 0.
//       Converted 9.113E-16 to 0.
//       Converted 103.919 to 104.
//       Converted 17834.191 to 17834.
//       1.79769313486232E+308 is outside the range of the Int16 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Decimal)

Converts the value of the specified decimal number to an equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Int16

`value`, rounded to the nearest 16-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue or less than Int16.MinValue.

Examples

The following example attempts to convert each element in an array of Decimal values to a 16-bit signed integer. The example illustrates that any fractional part of a Decimal value is rounded when performing the conversion.

C#

```
decimal[] values = { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                             9214.16m, Decimal.MaxValue };
short result;

foreach (decimal value in values)
{
   try {
      result = Convert.ToInt16(value);
      Console.WriteLine("Converted {0} to {1}.", value, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("{0} is outside the range of the Int16 type.",
                        value);
   }
}
// The example displays the following output:
//    -79228162514264337593543950335 is outside the range of the Int16
type.
//    Converted -1034.23 to -1034.
//    Converted -12 to -12.
//    Converted 0 to 0.
//    Converted 147 to 147.
//    Converted 9214.16 to 9214.
//    79228162514264337593543950335 is outside the range of the Int16
type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static short ToInt16 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Int16

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToInt16(Char)

Converts the value of the specified Unicode character to the equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is greater than Int16.MaxValue.

Examples

The following example attempts to convert each element in an array of Char values to a 16-bit signed integer.

C#

```
char[] chars = { 'a', 'z', '\x0007', '\x03FF',
                 '\x7FFF', '\xFFFE' };
short result;

foreach (char ch in chars)
{
   try {
      result = Convert.ToInt16(ch);
      Console.WriteLine("'{0}' converts to {1}.", ch, result);
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to convert u+{0} to an Int16.",
                        ((int)ch).ToString("X4"));
   }
}
// The example displays the following output:
//       'a' converts to 97.
//       'z' converts to 122.
//       '•' converts to 7.
//       'Ϗ' converts to 1023.
//       '翻' converts to 32767.
//       Unable to convert u+FFFE to an Int16.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of Byte values to a 16-bit signed integer.

C#

```
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
short result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToInt16(byteValue);
   Console.WriteLine("The Byte value {0} converts to {1}.",
                     byteValue, result);
}
// The example displays the following output:
//       The Byte value 0 converts to 0.
//       The Byte value 14 converts to 14.
//       The Byte value 122 converts to 122.
//       The Byte value 255 converts to 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Boolean)

Converts the specified Boolean value to the equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Int16

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example converts the Boolean values `true` and `false` to Int16 values.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToInt16(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToInt16(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt16(Object)

Converts the value of the specified object to a 16-bit signed integer.

C#

```
public static short ToInt16 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Int16

A 16-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format for an Int16 type.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example attempts to convert each element in an object array to a 16-bit signed integer.

C#

```
object[] values= { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                   "104", "103.0", "-1", "1.00e2", "One", 1.00e2};
short result;

foreach (object value in values)
{
   try {
      result = Convert.ToInt16(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int16 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to an Int16 exists for the {0}
value {1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value True to the Int16 value 1.
//    Converted the Int32 value -12 to the Int16 value -12.
//    Converted the Int32 value 163 to the Int16 value 163.
//    Converted the Int32 value 935 to the Int16 value 935.
//    Converted the Char value x to the Int16 value 120.
```

```
//    No conversion to an Int16 exists for the DateTime value 5/12/2009
12:00:00 AM.
//    Converted the String value 104 to the Int16 value 104.
//    The String value 103.0 is not in a recognizable format.
//    Converted the String value -1 to the Int16 value -1.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the Int16 value 100.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToInt16 implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 32-bit signed integer.

Overloads

Expand table

ToInt32(String)	Converts the specified string representation of a number to an equivalent 32-bit signed integer.
ToInt32(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit signed integer.
ToInt32(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 32-bit signed integer.
ToInt32(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 32-bit signed integer.
ToInt32(Object, IFormatProvider)	Converts the value of the specified object to a 32-bit signed integer, using the specified culture-specific formatting information.
ToInt32(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 32-bit signed integer, using the specified culture-specific formatting information.
ToInt32(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 32-bit signed integer.
ToInt32(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit signed integer.
ToInt32(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 32-bit signed integer.
ToInt32(Object)	Converts the value of the specified object to a 32-bit signed integer.
ToInt32(Int64)	Converts the value of the specified 64-bit signed integer to an

	equivalent 32-bit signed integer.
ToInt32(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit signed integer.
ToInt32(Char)	Converts the value of the specified Unicode character to the equivalent 32-bit signed integer.
ToInt32(DateTime)	Calling this method always throws InvalidCastException.
ToInt32(Boolean)	Converts the specified Boolean value to the equivalent 32-bit signed integer.
ToInt32(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 32-bit signed integer.
ToInt32(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 32-bit signed integer.
ToInt32(Int32)	Returns the specified 32-bit signed integer; no actual conversion is performed.
ToInt32(Decimal)	Converts the value of the specified decimal number to an equivalent 32-bit signed integer.

ToInt32(String)

Converts the specified string representation of a number to an equivalent 32-bit signed integer.

```
C#

public static int ToInt32 (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example attempts to convert each element in a numeric string array to an integer.

C#

```
string[] values = { "One", "1.34e28", "-26.87", "-18", "-6.00",
                    " 0", "137", "1601.9", Int32.MaxValue.ToString() };
int result;

foreach (string value in values)
{
   try {
      result = Convert.ToInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int32 type.",
value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value '{1}' is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    The String value 'One' is not in a recognizable format.
//    The String value '1.34e28' is not in a recognizable format.
//    The String value '-26.87' is not in a recognizable format.
//    Converted the String value '-18' to the Int32 value -18.
//    The String value '-6.00' is not in a recognizable format.
//    Converted the String value ' 0' to the Int32 value 0.
//    Converted the String value '137' to the Int32 value 137.
//    The String value '1601.9' is not in a recognizable format.
```

```
//    Converted the String value '2147483647' to the Int32 value
2147483647.
```

Remarks

Using the ToInt32(String) method is equivalent to passing `value` to the Int32.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Int32.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static int ToInt32 (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of 16-bit unsigned integers to an integer.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
int result;
foreach (ushort number in numbers)
{
   try {
      result = Convert.ToInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the Int32 value 0.
//    Converted the UInt16 value 121 to the Int32 value 121.
//    Converted the UInt16 value 340 to the Int32 value 340.
//    Converted the UInt16 value 65535 to the Int32 value 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 32-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static int ToInt32 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int32.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned integers to a signed integer.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
int result;
foreach (uint number in numbers)
{
   try {
      result = Convert.ToInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt32 value 0 to the Int32 value 0.
//    Converted the UInt32 value 121 to the Int32 value 121.
//    Converted the UInt32 value 340 to the Int32 value 340.
//    The UInt32 value 4294967295 is outside the range of the Int32 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Single)

Converts the value of the specified single-precision floating-point number to an equivalent 32-bit signed integer.

```
C#

public static int ToInt32 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Int32

`value`, rounded to the nearest 32-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int32.MaxValue or less than Int32.MinValue.

Examples

The following example attempts to convert each element in an array of Single values to an integer.

```
C#

float[] values= { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                  0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
int result;

foreach (float value in values)
{
   try {
      result = Convert.ToInt32(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
```

```
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int32 type.",
value);
   }
}
// The example displays the following output:
//    -3.40282346638529E+38 is outside the range of the Int32 type.
//    -13799999488 is outside the range of the Int32 type.
//    Converted the Double value -1023.29901123047 to the Int32 value
-1023.
//    Converted the Double value -12.9799995422363 to the Int32 value
-13.
//    Converted the Double value 0 to the Int32 value 0.
//    Converted the Double value 9.11299983940444E-16 to the Int32 value
0.
//    Converted the Double value 103.918998718262 to the Int32 value 104.
//    Converted the Double value 17834.19140625 to the Int32 value 17834.
//    3.40282346638529E+38 is outside the range of the Int32 type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Object, IFormatProvider)

Converts the value of the specified object to a 32-bit signed integer, using the specified culture-specific formatting information.

```
C#

public static int ToInt32 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement IConvertible.

OverflowException

`value` represents a number that is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the `ToInt32` method. This method returns an average of the array of Double values, using the object that implements IFormatProvider to determine how to calculate the average.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }

    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;
```

```
    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
        {
            if( index >= 0 && index < data.Count )
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;

        if( data.Count == 0 )
            return 0.0;
```

```
        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];

                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];

            default:
                return 0.0;
        }
    }

    // Get the AverageInfo object from the caller's format provider,
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
```

```
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }

    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
            GetAverageInfo( provider ).TypeOfAverage )
        {
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
            Byte.MinValue, Byte.MaxValue, provider ) );
    }
```

```
    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
    }

    public double ToDouble( IFormatProvider provider )
    {
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
```

```
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }

    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}

class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
```

```
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );

         // Display the dataset elements.
        if( ds.Count > 0 )
        {
            Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
            for( int iX = 1; iX < ds.Count; iX++ )
                Console.Write( ", {0}", ds[ iX ] );
            Console.WriteLine( "]\n" );
        }

        Console.WriteLine( fmt, "Convert.", "Default",
            "Geometric Mean", "Median");
        Console.WriteLine( fmt, "--------", "-------",
            "--------------", "------");
        Console.WriteLine( fmt, "ToBoolean",
            Convert.ToBoolean( ds, null ),
            Convert.ToBoolean( ds, geMean ),
            Convert.ToBoolean( ds, median ) );
        Console.WriteLine( fmt, "ToByte",
            Convert.ToByte( ds, null ),
            Convert.ToByte( ds, geMean ),
            Convert.ToByte( ds, median ) );
        Console.WriteLine( fmt, "ToChar",
            Convert.ToChar( ds, null ),
            Convert.ToChar( ds, geMean ),
            Convert.ToChar( ds, median ) );
        Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
            "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
            "ToDateTime", Convert.ToDateTime( ds, null ),
            Convert.ToDateTime( ds, geMean ),
            Convert.ToDateTime( ds, median ) );
        Console.WriteLine( fmt, "ToDecimal",
            Convert.ToDecimal( ds, null ),
            Convert.ToDecimal( ds, geMean ),
            Convert.ToDecimal( ds, median ) );
        Console.WriteLine( fmt, "ToDouble",
            Convert.ToDouble( ds, null ),
            Convert.ToDouble( ds, geMean ),
            Convert.ToDouble( ds, median ) );
        Console.WriteLine( fmt, "ToInt16",
            Convert.ToInt16( ds, null ),
            Convert.ToInt16( ds, geMean ),
            Convert.ToInt16( ds, median ) );
        Console.WriteLine( fmt, "ToInt32",
            Convert.ToInt32( ds, null ),
            Convert.ToInt32( ds, geMean ),
            Convert.ToInt32( ds, median ) );
        Console.WriteLine( fmt, "ToInt64",
            Convert.ToInt64( ds, null ),
```

```
            Convert.ToInt64( ds, geMean ),
            Convert.ToInt64( ds, median ) );
        Console.WriteLine( fmt, "ToSByte",
            Convert.ToSByte( ds, null ),
            Convert.ToSByte( ds, geMean ),
            Convert.ToSByte( ds, median ) );
        Console.WriteLine( fmt, "ToSingle",
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );

        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );

        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       103                  59                  67
ToChar                         g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
```

```
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                      103                  59                  67
ToInt32                      103                  59                  67
ToInt64                      103                  59                  67
ToSByte                      103                  59                  67
ToSingle                103.4917            59.43321                67.3
ToUInt16                     103                  59                  67
ToUInt32                     103                  59                  67
ToUInt64                     103                  59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal              750999.89    631577.237188435            499999.5
ToDouble               750999.89    631577.237188435            499999.5
ToInt16                    32767               32767               32767
ToInt32                   751000              631577              500000
ToInt64                   751000              631577              500000
ToSByte                      127                 127                 127
ToSingle                750999.9            631577.3            499999.5
ToUInt16                   65535               65535               65535
ToUInt32                  751000              631577              500000
ToUInt64                  751000              631577              500000
*/
```

Remarks

The return value is the result of invoking the IConvertible.ToInt32 method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent 32-bit signed integer, using the specified culture-specific formatting information.

C#

```
public static int ToInt32 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int32

A 32-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that recognizes the string "pos" as the positive sign and the string "neg" as the negative sign. It then attempts to convert each element of a numeric string array to an integer using both this provider and the NumberFormatInfo provider for the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a custom NumberFormatInfo object and set its two
properties
      // used by default in parsing numeric strings.
      NumberFormatInfo customProvider = new NumberFormatInfo();
      customProvider.NegativeSign = "neg ";
      customProvider.PositiveSign = "pos ";

      // Add custom and invariant provider to an array of providers.
      NumberFormatInfo[] providers = { customProvider,
NumberFormatInfo.InvariantInfo };

      // Define an array of strings to convert.
      string[] numericStrings = { "123456789", "+123456789", "pos
123456789",
                                  "-123456789", "neg 123456789",
"123456789.",
                                  "123,456,789", "(123456789)",
"2147483648",
                                  "-2147483649" };

      // Use each provider to parse all the numeric strings.
      for (int ctr = 0; ctr <= 1; ctr++)
      {
         IFormatProvider provider = providers[ctr];
         Console.WriteLine(ctr == 0 ? "Custom Provider:" : "Invariant
Provider:");
         foreach (string numericString in numericStrings)
         {
            Console.Write("{0,15}  --> ", numericString);
            try {
               Console.WriteLine("{0,20}", Convert.ToInt32(numericString,
provider));
```

```
            }
            catch (FormatException) {
               Console.WriteLine("{0,20}", "FormatException");
            }
            catch (OverflowException) {
               Console.WriteLine("{0,20}", "OverflowException");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       Custom Provider:
//             123456789  -->            123456789
//            +123456789  -->      FormatException
//         pos 123456789  -->            123456789
//            -123456789  -->      FormatException
//         neg 123456789  -->           -123456789
//            123456789.  -->      FormatException
//           123,456,789  -->      FormatException
//           (123456789)  -->      FormatException
//            2147483648  -->    OverflowException
//           -2147483649  -->      FormatException
//
//       Invariant Provider:
//             123456789  -->            123456789
//            +123456789  -->            123456789
//         pos 123456789  -->      FormatException
//            -123456789  -->           -123456789
//         neg 123456789  -->      FormatException
//            123456789.  -->      FormatException
//           123,456,789  -->      FormatException
//           (123456789)  -->      FormatException
//            2147483648  -->    OverflowException
//           -2147483649  -->    OverflowException
```

Remarks

The return value is the result of invoking the Int32.Parse method on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Int32.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(String, Int32)

Converts the string representation of a number in a specified base to an equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

Int32

A 32-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

ArgumentOutOfRangeException
`value` is Empty.

FormatException
`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException
`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than Int32.MinValue or greater than Int32.MaxValue.

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the negative sign is not supported for non-base 10 numeric representations, the ToInt32(String, Int32) method assumes that negative numbers use two's complement representation. In other words, the method always interprets the highest-order binary bit of an integer (bit 31) as its sign bit. As a result, it is possible to write code in which a non-base 10 number that is out of the range of the Int32 data type is converted to an Int32 value without the method throwing an exception. The following example increments Int32.MaxValue by one, converts the resulting number to its hexadecimal string representation, and then calls the ToInt32(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x80000000 converts to -2147483648."

```
C#
```

```
// Create a hexadecimal value out of range of the Integer type.
string value = Convert.ToString((long) int.MaxValue + 1, 16);
// Convert it back to a number.
try
{
   int number = Convert.ToInt32(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number.ToString());
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an integer.", value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method is using the appropriate numeric representation to interpret a particular value. As the following example illustrates, you can ensure that the method handles overflows appropriately by first retrieving the sign of the numeric value before converting it to its hexadecimal string representation. Throw an exception if the original value was positive but the conversion back to an integer yields a negative value.

C#

```
// Create a hexadecimal value out of range of the Integer type.
long sourceNumber = (long) int.MaxValue + 1;
bool isNegative = Math.Sign(sourceNumber) == -1;
string value = Convert.ToString(sourceNumber, 16);
int targetNumber;
try
{
   targetNumber = Convert.ToInt32(value, 16);
   if (!(isNegative) & (targetNumber & 0x80000000) != 0)
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an integer.", value);
}
// Displays the following to the console:
//    Unable to convert '0x80000000' to an integer.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static int ToInt32 (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int32.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned long integers to a signed integer.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
int result;
foreach (ulong number in numbers)
{
   try {
      result = Convert.ToInt32(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt64 value 0 to a Int32 value 0.
//    Converted the UInt64 value 121 to a Int32 value 121.
//    Converted the UInt64 value 340 to a Int32 value 340.
//    The UInt64 value 18446744073709551615 is outside the range of the
Int32 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 32-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static int ToInt32 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Int32

A 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of signed bytes to an integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
int result;

foreach (sbyte number in numbers)
{
   result = Convert.ToInt32(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
```

```
}
// The example displays the following output:
//       Converted the SByte value -128 to the Int32 value -128.
//       Converted the SByte value -1 to the Int32 value -1.
//       Converted the SByte value 0 to the Int32 value 0.
//       Converted the SByte value 10 to the Int32 value 10.
//       Converted the SByte value 127 to the Int32 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Object)

Converts the value of the specified object to a 32-bit signed integer.

C#

```
public static int ToInt32 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Int32

A 32-bit signed integer equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example attempts to convert each element in an object array to an integer.

C#

```
object[] values = { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                    "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2, 16.3e42};
int result;

foreach (object value in values)
{
   try {
      result = Convert.ToInt32(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int32 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to an Int32 exists for the {0}
```

```
value {1}.",
                        value.GetType().Name, value);
    }
}
// The example displays the following output:
//    Converted the Boolean value True to the Int32 value 1.
//    Converted the Int32 value -12 to the Int32 value -12.
//    Converted the Int32 value 163 to the Int32 value 163.
//    Converted the Int32 value 935 to the Int32 value 935.
//    Converted the Char value x to the Int32 value 120.
//    No conversion to an Int32 exists for the DateTime value 5/12/2009
12:00:00 AM.
//    Converted the String value 104 to the Int32 value 104.
//    The String value 103.0 is not in a recognizable format.
//    Converted the String value -1 to the Int32 value -1.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the Int32 value 100.
//    The Double value 1.63E+43 is outside the range of the Int32 type.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToInt32 implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Exceptions

OverflowException

`value` is greater than Int32.MaxValue or less than Int32.MinValue.

Examples

The following example attempts to convert each element in an array of long integers to an integer.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
int result;
foreach (long number in numbers)
{
   try {
      result = Convert.ToInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
Int32 type.
```

```
//    Converted the Int64 value -1 to the Int32 value -1.
//    Converted the Int64 value 0 to the Int32 value 0.
//    Converted the Int64 value 121 to the Int32 value 121.
//    Converted the Int64 value 340 to the Int32 value 340.
//    The Int64 value 9223372036854775807 is outside the range of the
Int32 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of bytes to an integer.

C#

```
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
int result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToInt32(byteValue);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     byteValue.GetType().Name, byteValue,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the Byte value 0 to the Int32 value 0.
//       Converted the Byte value 14 to the Int32 value 14.
//       Converted the Byte value 122 to the Int32 value 122.
//       Converted the Byte value 255 to the Int32 value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Char)

Converts the value of the specified Unicode character to the equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of Char values to an integer.

C#

```
char[] chars = { 'a', 'z', '\u0007', '\u03FF',
                 '\u7FFF', '\uFFFE' };
int result;

foreach (char ch in chars)
{
   try {
      result = Convert.ToInt32(ch);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        ch.GetType().Name, ch,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to convert u+{0} to an Int32.",
                        ((int)ch).ToString("X4"));
   }
}
// The example displays the following output:
//       Converted the Char value 'a' to the Int32 value 97.
//       Converted the Char value 'z' to the Int32 value 122.
//       Converted the Char value '' to the Int32 value 7.
//       Converted the Char value 'Ϗ' to the Int32 value 1023.
//       Converted the Char value '翻' to the Int32 value 32767.
//       Converted the Char value '￾' to the Int32 value 65534.
```

Remarks

The ToInt32(Char) method returns a 32-bit signed integer that represents the UTF-16 encoded code unit of the `value` argument. If `value` is not a low surrogate or a high

surrogate, this return value also represents the Unicode code point of value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static int ToInt32 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Int32

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToInt32(Boolean)

Converts the specified Boolean value to the equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Int32

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example converts the Boolean values `true` and `false` to integers.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToInt32(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToInt32(trueFlag));
// The example displays the following output:
```

```
//        False converts to 0.
//        True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Double)

Converts the value of the specified double-precision floating-point number to an equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Int32

`value`, rounded to the nearest 32-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int32.MaxValue or less than Int32.MinValue.

Examples

The following example attempts to convert each element in an array of Double values to an integer.

C#

```
double[] values= { Double.MinValue, -1.38e10, -1023.299, -12.98,
                   0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
int result;

foreach (double value in values)
{
   try {
      result = Convert.ToInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int32 type.",
value);
   }
}
//    -1.79769313486232E+308 is outside the range of the Int32 type.
//    -13800000000 is outside the range of the Int32 type.
//    Converted the Double value '-1023.299' to the Int32 value -1023.
//    Converted the Double value '-12.98' to the Int32 value -13.
//    Converted the Double value '0' to the Int32 value 0.
//    Converted the Double value '9.113E-16' to the Int32 value 0.
//    Converted the Double value '103.919' to the Int32 value 104.
//    Converted the Double value '17834.191' to the Int32 value 17834.
//    1.79769313486232E+308 is outside the range of the Int32 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Int32

A 32-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of 16-bit signed integers to an integer.

C#

```
short[] numbers= { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
int result;

foreach (short number in numbers)
{
```

```
    result = Convert.ToInt32(number);
    Console.WriteLine("Converted the {0} value {1} to a {2} value {3}.",
                      number.GetType().Name, number,
                      result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int16 value -32768 to a Int32 value -32768.
//    Converted the Int16 value -1 to a Int32 value -1.
//    Converted the Int16 value 0 to a Int32 value 0.
//    Converted the Int16 value 121 to a Int32 value 121.
//    Converted the Int16 value 340 to a Int32 value 340.
//    Converted the Int16 value 32767 to a Int32 value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Int32)

Returns the specified 32-bit signed integer; no actual conversion is performed.

C#

```
public static int ToInt32 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to return.

Returns

Int32

`value` is returned unchanged.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt32(Decimal)

Converts the value of the specified decimal number to an equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Int32

`value`, rounded to the nearest 32-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int32.MaxValue or less than Int32.MinValue.

Examples

The following example attempts to convert each element in an array of Decimal values to an integer.

C#

```
decimal[] values= { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                    199.55m, 9214.16m, Decimal.MaxValue };
int result;

foreach (decimal value in values)
{
   try {
      result = Convert.ToInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int32 type.",
                        value);
   }
}
// The example displays the following output:
//    -79228162514264337593543950335 is outside the range of the Int32
type.
//    Converted the Decimal value '-1034.23' to the Int32 value -1034.
//    Converted the Decimal value '-12' to the Int32 value -12.
//    Converted the Decimal value '0' to the Int32 value 0.
//    Converted the Decimal value '147' to the Int32 value 147.
//    Converted the Decimal value '199.55' to the Int32 value 200.
//    Converted the Decimal value '9214.16' to the Int32 value 9214.
//    79228162514264337593543950335 is outside the range of the Int32
type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 64-bit signed integer.

Overloads

Expand table

ToInt64(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 64-bit signed integer.
ToInt64(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 64-bit signed integer.
ToInt64(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 64-bit signed integer, using the specified culture-specific formatting information.
ToInt64(Object, IFormatProvider)	Converts the value of the specified object to a 64-bit signed integer, using the specified culture-specific formatting information.
ToInt64(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 64-bit signed integer.
ToInt64(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit signed integer.
ToInt64(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit signed integer.
ToInt64(String)	Converts the specified string representation of a number to an equivalent 64-bit signed integer.
ToInt64(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 64-bit signed integer.
ToInt64(Object)	Converts the value of the specified object to a 64-bit signed integer.
ToInt64(Int64)	Returns the specified 64-bit signed integer; no actual conversion is

	performed.
ToInt64(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit signed integer.
ToInt64(Char)	Converts the value of the specified Unicode character to the equivalent 64-bit signed integer.
ToInt64(DateTime)	Calling this method always throws InvalidCastException.
ToInt64(Boolean)	Converts the specified Boolean value to the equivalent 64-bit signed integer.
ToInt64(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 64-bit signed integer.
ToInt64(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent 64-bit signed integer.
ToInt64(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 64-bit signed integer.
ToInt64(Decimal)	Converts the value of the specified decimal number to an equivalent 64-bit signed integer.

ToInt64(Single)

Converts the value of the specified single-precision floating-point number to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Int64

`value`, rounded to the nearest 64-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int64.MaxValue or less than Int64.MinValue.

Examples

The following example attempts to convert each element in an array of Single values to a long integer.

C#

```
float[] values= { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                  0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
long result;

foreach (float value in values)
{
   try {
      result = Convert.ToInt64(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int64 type.",
value);
   }
}
// The example displays the following output:
//    -3.4028235E+38 is outside the range of the Int64 type.
//    Converted the Single value -1.38E+10 to the Int64 value
-13799999488.
//    Converted the Single value -1023.299 to the Int64 value -1023.
//    Converted the Single value -12.98 to the Int64 value -13.
//    Converted the Single value 0 to the Int64 value 0.
//    Converted the Single value 9.113E-16 to the Int64 value 0.
//    Converted the Single value 103.919 to the Int64 value 104.
//    Converted the Single value 17834.191 to the Int64 value 17834.
//    3.4028235E+38 is outside the range of the Int64 type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(String, Int32)

Converts the string representation of a number in a specified base to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

Int64

A 64-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

ArgumentOutOfRangeException
`value` is Empty.

FormatException
`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException
`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than Int64.MinValue or greater than Int64.MaxValue.

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the negative sign is not supported for non-base 10 numeric representations, the ToInt64(String, Int32) method assumes that negative numbers use two's complement representation. In other words, the method always interprets the highest-order binary bit of a long integer (bit 63) as its sign bit. As a result, it is possible to write code in which a non-base 10 number that is out of the range of the Int64 data type is converted to an Int64 value without the method throwing an exception. The following example converts MaxValue to its hexadecimal string representation, and then calls the ToInt64(String, Int32) method. Instead of throwing an exception, the method displays the message, "0xFFFFFFFFFFFFFFFF converts to -1."

```
C#
```

```
// Create a hexadecimal value out of range of the long type.
string value = ulong.MaxValue.ToString("X");
// Use Convert.ToInt64 to convert it back to a number.
try
{
   long number = Convert.ToInt64(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to a long integer.",
value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method is using the appropriate numeric representation to interpret a particular value. As the following example illustrates, you can ensure that the method handles overflows appropriately by first determining whether a value represents an unsigned or a signed type when converting it to its hexadecimal string representation. Throw an exception if the original value was an unsigned type but the conversion back to an integer yields a value whose sign bit is on.

C#

```
// Create a negative hexadecimal value out of range of the Byte type.
ulong sourceNumber = ulong.MaxValue;
bool isSigned =
Math.Sign(Convert.ToDouble(sourceNumber.GetType().GetField("MinValue").Ge
tValue(null))) == -1;
string value = sourceNumber.ToString("X");
long targetNumber;
try
{
   targetNumber = Convert.ToInt64(value, 16);
   if (! isSigned && ((targetNumber & 0x80000000) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to a long integer.",
value);
}
// Displays the following to the console:
//    Unable to convert '0xFFFFFFFFFFFFFFFF' to a long integer.
```

The following example attempts to interpret each element in a string array as a hexadecimal string and convert it to a long integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] hexStrings = { "8000000000000000", "0FFFFFFFFFFFFFFF",
                              "f0000000000001000", "00A30", "D", "-13",
"GAD" };
      foreach (string hexString in hexStrings)
      {
         try {
            long number = Convert.ToInt64(hexString, 16);
            Console.WriteLine("Converted '{0}' to {1:N0}.", hexString,
number);
         }
         catch (FormatException) {
            Console.WriteLine("'{0}' is not in the correct format for a
hexadecimal number.",
                              hexString);
         }
         catch (OverflowException) {
            Console.WriteLine("'{0}' is outside the range of an Int64.",
hexString);
         }
         catch (ArgumentException) {
            Console.WriteLine("'{0}' is invalid in base 16.", hexString);
         }
      }
   }
}
// The example displays the following output:
//       Converted '8000000000000000' to -9,223,372,036,854,775,808.
//       Converted '0FFFFFFFFFFFFFFF' to 1,152,921,504,606,846,975.
//       'f0000000000001000' is outside the range of an Int64.
//       Converted '00A30' to 2,608.
//       Converted 'D' to 13.
//       '-13' is invalid in base 16.
//       'GAD' is not in the correct format for a hexadecimal number.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent 64-bit signed integer, using the specified culture-specific formatting information.

C#

```
public static long ToInt64 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int64

A 64-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that recognizes the string "pos" as the positive sign and the string "neg" as the negative sign. It then attempts to convert each element of a numeric string array to an integer using both this provider and the NumberFormatInfo provider for the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set the properties that
      // affect conversions using Convert.ToInt64(String,
IFormatProvider).
      NumberFormatInfo customProvider = new NumberFormatInfo();
      customProvider.NegativeSign = "neg ";
      customProvider.PositiveSign = "pos ";

      // Create an array of providers with the custom provider and the
      // NumberFormatInfo object for the invariant culture.
      NumberFormatInfo[] providers = { customProvider,
                                       NumberFormatInfo.InvariantInfo };

      // Define an array of strings to parse.
      string[] numericStrings = { "123456789", "+123456789", "pos
123456789",
                                  "-123456789", "neg 123456789",
"123456789.",
                                  "123,456,789", "(123456789)",
                                  "9223372036854775808",
"-9223372036854775809" };

      for (int ctr = 0; ctr < 2; ctr++)
      {
         IFormatProvider provider = providers[ctr];
         Console.WriteLine(ctr == 0 ? "Custom Provider:" : "Invariant
Culture:");
         foreach (string numericString in numericStrings)
         {
            Console.Write("   {0,-22} -->  ", numericString);
            try {
               Console.WriteLine("{0,22}", Convert.ToInt32(numericString,
provider));
```

```
            }
            catch (FormatException) {
               Console.WriteLine("{0,22}", "Unrecognized Format");
            }
            catch (OverflowException) {
               Console.WriteLine("{0,22}", "Overflow");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       Custom Provider:
//          123456789              -->               123456789
//          +123456789             -->     Unrecognized Format
//          pos 123456789          -->               123456789
//          -123456789             -->     Unrecognized Format
//          neg 123456789          -->              -123456789
//          123456789.             -->     Unrecognized Format
//          123,456,789            -->     Unrecognized Format
//          (123456789)            -->     Unrecognized Format
//          9223372036854775808    -->                Overflow
//          -9223372036854775809   -->     Unrecognized Format
//
//       Invariant Culture:
//          123456789              -->               123456789
//          +123456789             -->               123456789
//          pos 123456789          -->     Unrecognized Format
//          -123456789             -->              -123456789
//          neg 123456789          -->     Unrecognized Format
//          123456789.             -->     Unrecognized Format
//          123,456,789            -->     Unrecognized Format
//          (123456789)            -->     Unrecognized Format
//          9223372036854775808    -->                Overflow
//          -9223372036854775809   -->                Overflow
```

Remarks

The return value is the result of invoking the Int64.Parse method on `value`.

`provider` is an `IFormatProvider` instance that obtains a NumberFormatInfo object. The `NumberFormatInfo` object provides culture-specific information about the format of `value`. If `provider` is `null`, the `NumberFormatInfo` for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Int64.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Object, IFormatProvider)

Converts the value of the specified object to a 64-bit signed integer, using the specified culture-specific formatting information.

C#

```
public static long ToInt64 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example defines a class that implements IConvertible and a class that implements IFormatProvider. Objects of the class that implements IConvertible hold an array of Double values. An object of each class is passed to the `ToInt64` method. This method returns an average of the array of Double values, using the object that implements IFormatProvider to determine how to calculate the average.

C#

```
using System;
using System.Collections;

// Define the types of averaging available in the class
// implementing IConvertible.
public enum AverageType : short
{
    None = 0,
    GeometricMean = 1,
    ArithmeticMean = 2,
    Median = 3
};

// Pass an instance of this class to methods that require an
// IFormatProvider. The class instance determines the type of
// average to calculate.
public class AverageInfo : IFormatProvider
{
    protected AverageType AvgType;

    // Specify the type of averaging in the constructor.
    public AverageInfo( AverageType avgType )
    {
        this.AvgType = avgType;
    }

    // This method returns a reference to the containing object
    // if an object of AverageInfo type is requested.
```

```
    public object GetFormat( Type argType )
    {
        if ( argType == typeof( AverageInfo ) )
            return this;
        else
            return null;
    }

    // Use this property to set or get the type of averaging.
    public AverageType TypeOfAverage
    {
        get { return this.AvgType; }
        set { this.AvgType = value; }
    }
}

// This class encapsulates an array of double values and implements
// the IConvertible interface. Most of the IConvertible methods
// return an average of the array elements in one of three types:
// arithmetic mean, geometric mean, or median.
public class DataSet : IConvertible
{
    protected ArrayList     data;
    protected AverageInfo   defaultProvider;

    // Construct the object and add an initial list of values.
    // Create a default format provider.
    public DataSet( params double[ ] values )
    {
        data = new ArrayList( values );
        defaultProvider =
            new AverageInfo( AverageType.ArithmeticMean );
    }

    // Add additional values with this method.
    public int Add( double value )
    {
        data.Add( value );
        return data.Count;
    }

    // Get, set, and add values with this indexer property.
    public double this[ int index ]
    {
        get
        {
            if( index >= 0 && index < data.Count )
                return (double)data[ index ];
            else
                throw new InvalidOperationException(
                    "[DataSet.get] Index out of range." );
        }
        set
        {
            if( index >= 0 && index < data.Count )
```

```
                data[ index ] = value;

            else if( index == data.Count )
                data.Add( value );
            else
                throw new InvalidOperationException(
                    "[DataSet.set] Index out of range." );
        }
    }

    // This property returns the number of elements in the object.
    public int Count
    {
        get { return data.Count; }
    }

    // This method calculates the average of the object's elements.
    protected double Average( AverageType avgType )
    {
        double  SumProd;

        if( data.Count == 0 )
            return 0.0;

        switch( avgType )
        {
            case AverageType.GeometricMean:

                SumProd = 1.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd *= (double)data[ Index ];

                // This calculation will not fail with negative
                // elements.
                return Math.Sign( SumProd ) * Math.Pow(
                    Math.Abs( SumProd ), 1.0 / data.Count );

            case AverageType.ArithmeticMean:

                SumProd = 0.0;
                for( int Index = 0; Index < data.Count; Index++ )
                    SumProd += (double)data[ Index ];

                return SumProd / data.Count;

            case AverageType.Median:

                if( data.Count % 2 == 0 )
                    return ( (double)data[ data.Count / 2 ] +
                        (double)data[ data.Count / 2 - 1 ] ) / 2.0;
                else
                    return (double)data[ data.Count / 2 ];

            default:
                return 0.0;
```

```
        }
    }

    // Get the AverageInfo object from the caller's format provider,
    // or use the local default.
    protected AverageInfo GetAverageInfo( IFormatProvider provider )
    {
        AverageInfo avgInfo = null;

        if( provider != null )
            avgInfo = (AverageInfo)provider.GetFormat(
                typeof( AverageInfo ) );

        if ( avgInfo == null )
            return defaultProvider;
        else
            return avgInfo;
    }

    // Calculate the average and limit the range.
    protected double CalcNLimitAverage( double min, double max,
        IFormatProvider provider )
    {
        // Get the format provider and calculate the average.
        AverageInfo avgInfo = GetAverageInfo( provider );
        double avg = Average( avgInfo.TypeOfAverage );

        // Limit the range, based on the minimum and maximum values
        // for the type.
        return avg > max ? max : avg < min ? min : avg;
    }

    // The following elements are required by IConvertible.

    // None of these conversion functions throw exceptions. When
    // the data is out of range for the type, the appropriate
    // MinValue or MaxValue is used.
    public TypeCode GetTypeCode( )
    {
        return TypeCode.Object;
    }

    public bool ToBoolean( IFormatProvider provider )
    {
        // ToBoolean is false if the dataset is empty.
        if( data.Count <= 0 )
        {
            return false;
        }

        // For median averaging, ToBoolean is true if any
        // non-discarded elements are nonzero.
        else if( AverageType.Median ==
            GetAverageInfo( provider ).TypeOfAverage )
        {
```

```
            if (data.Count % 2 == 0 )
                return ( (double)data[ data.Count / 2 ] != 0.0 ||
                    (double)data[ data.Count / 2 - 1 ] != 0.0 );
            else
                return (double)data[ data.Count / 2 ] != 0.0;
        }

        // For arithmetic or geometric mean averaging, ToBoolean is
        // true if any element of the dataset is nonzero.
        else
        {
            for( int Index = 0; Index < data.Count; Index++ )
                if( (double)data[ Index ] != 0.0 )
                    return true;
            return false;
        }
    }

    public byte ToByte( IFormatProvider provider )
    {
        return Convert.ToByte( CalcNLimitAverage(
            Byte.MinValue, Byte.MaxValue, provider ) );
    }

    public char ToChar( IFormatProvider provider )
    {
        return Convert.ToChar( Convert.ToUInt16( CalcNLimitAverage(
            Char.MinValue, Char.MaxValue, provider ) ) );
    }

    // Convert to DateTime by adding the calculated average as
    // seconds to the current date and time. A valid DateTime is
    // always returned.
    public DateTime ToDateTime( IFormatProvider provider )
    {
        double seconds =
            Average( GetAverageInfo( provider ).TypeOfAverage );
        try
        {
            return DateTime.Now.AddSeconds( seconds );
        }
        catch( ArgumentOutOfRangeException )
        {
            return seconds < 0.0 ? DateTime.MinValue : DateTime.MaxValue;
        }
    }

    public decimal ToDecimal( IFormatProvider provider )
    {
        // The Double conversion rounds Decimal.MinValue and
        // Decimal.MaxValue to invalid Decimal values, so the
        // following limits must be used.
        return Convert.ToDecimal( CalcNLimitAverage(
            -79228162514264330000000000000.0,
            79228162514264330000000000000.0, provider ) );
```

```
    }

    public double ToDouble( IFormatProvider provider )
    {
        return Average( GetAverageInfo(provider).TypeOfAverage );
    }

    public short ToInt16( IFormatProvider provider )
    {
        return Convert.ToInt16( CalcNLimitAverage(
            Int16.MinValue, Int16.MaxValue, provider ) );
    }

    public int ToInt32( IFormatProvider provider )
    {
        return Convert.ToInt32( CalcNLimitAverage(
            Int32.MinValue, Int32.MaxValue, provider ) );
    }

    public long ToInt64( IFormatProvider provider )
    {
        // The Double conversion rounds Int64.MinValue and
        // Int64.MaxValue to invalid Int64 values, so the following
        // limits must be used.
        return Convert.ToInt64( CalcNLimitAverage(
            -9223372036854775000, 9223372036854775000, provider ) );
    }

    public SByte ToSByte( IFormatProvider provider )
    {
        return Convert.ToSByte( CalcNLimitAverage(
            SByte.MinValue, SByte.MaxValue, provider ) );
    }

    public float ToSingle( IFormatProvider provider )
    {
        return Convert.ToSingle( CalcNLimitAverage(
            Single.MinValue, Single.MaxValue, provider ) );
    }

    public UInt16 ToUInt16( IFormatProvider provider )
    {
        return Convert.ToUInt16( CalcNLimitAverage(
            UInt16.MinValue, UInt16.MaxValue, provider ) );
    }

    public UInt32 ToUInt32( IFormatProvider provider )
    {
        return Convert.ToUInt32( CalcNLimitAverage(
            UInt32.MinValue, UInt32.MaxValue, provider ) );
    }

    public UInt64 ToUInt64( IFormatProvider provider )
    {
        // The Double conversion rounds UInt64.MaxValue to an invalid
```

```
        // UInt64 value, so the following limit must be used.
        return Convert.ToUInt64( CalcNLimitAverage(
            0, 18446744073709550000.0, provider ) );
    }

    public object ToType( Type conversionType,
        IFormatProvider provider )
    {
        return Convert.ChangeType( Average(
            GetAverageInfo( provider ).TypeOfAverage ),
            conversionType );
    }

    public string ToString( IFormatProvider provider )
    {
        AverageType avgType = GetAverageInfo( provider ).TypeOfAverage;
        return String.Format( "( {0}: {1:G10} )", avgType,
            Average( avgType ) );
    }
}

class IConvertibleProviderDemo
{
    // Display a DataSet with three different format providers.
    public static void DisplayDataSet( DataSet ds )
    {
        string      fmt    = "{0,-12}{1,20}{2,20}{3,20}";
        AverageInfo median = new AverageInfo( AverageType.Median );
        AverageInfo geMean =
            new AverageInfo( AverageType.GeometricMean );

         // Display the dataset elements.
        if( ds.Count > 0 )
        {
            Console.Write( "\nDataSet: [{0}", ds[ 0 ] );
            for( int iX = 1; iX < ds.Count; iX++ )
                Console.Write( ", {0}", ds[ iX ] );
            Console.WriteLine( "]\n" );
        }

        Console.WriteLine( fmt, "Convert.", "Default",
            "Geometric Mean", "Median");
        Console.WriteLine( fmt, "--------", "-------",
            "--------------", "------");
        Console.WriteLine( fmt, "ToBoolean",
            Convert.ToBoolean( ds, null ),
            Convert.ToBoolean( ds, geMean ),
            Convert.ToBoolean( ds, median ) );
        Console.WriteLine( fmt, "ToByte",
            Convert.ToByte( ds, null ),
            Convert.ToByte( ds, geMean ),
            Convert.ToByte( ds, median ) );
        Console.WriteLine( fmt, "ToChar",
            Convert.ToChar( ds, null ),
            Convert.ToChar( ds, geMean ),
```

```
            Convert.ToChar( ds, median ) );
        Console.WriteLine( "{0,-12}{1,20:yyyy-MM-dd HH:mm:ss}" +
            "{2,20:yyyy-MM-dd HH:mm:ss}{3,20:yyyy-MM-dd HH:mm:ss}",
            "ToDateTime", Convert.ToDateTime( ds, null ),
            Convert.ToDateTime( ds, geMean ),
            Convert.ToDateTime( ds, median ) );
        Console.WriteLine( fmt, "ToDecimal",
            Convert.ToDecimal( ds, null ),
            Convert.ToDecimal( ds, geMean ),
            Convert.ToDecimal( ds, median ) );
        Console.WriteLine( fmt, "ToDouble",
            Convert.ToDouble( ds, null ),
            Convert.ToDouble( ds, geMean ),
            Convert.ToDouble( ds, median ) );
        Console.WriteLine( fmt, "ToInt16",
            Convert.ToInt16( ds, null ),
            Convert.ToInt16( ds, geMean ),
            Convert.ToInt16( ds, median ) );
        Console.WriteLine( fmt, "ToInt32",
            Convert.ToInt32( ds, null ),
            Convert.ToInt32( ds, geMean ),
            Convert.ToInt32( ds, median ) );
        Console.WriteLine( fmt, "ToInt64",
            Convert.ToInt64( ds, null ),
            Convert.ToInt64( ds, geMean ),
            Convert.ToInt64( ds, median ) );
        Console.WriteLine( fmt, "ToSByte",
            Convert.ToSByte( ds, null ),
            Convert.ToSByte( ds, geMean ),
            Convert.ToSByte( ds, median ) );
        Console.WriteLine( fmt, "ToSingle",
            Convert.ToSingle( ds, null ),
            Convert.ToSingle( ds, geMean ),
            Convert.ToSingle( ds, median ) );
        Console.WriteLine( fmt, "ToUInt16",
            Convert.ToUInt16( ds, null ),
            Convert.ToUInt16( ds, geMean ),
            Convert.ToUInt16( ds, median ) );
        Console.WriteLine( fmt, "ToUInt32",
            Convert.ToUInt32( ds, null ),
            Convert.ToUInt32( ds, geMean ),
            Convert.ToUInt32( ds, median ) );
        Console.WriteLine( fmt, "ToUInt64",
            Convert.ToUInt64( ds, null ),
            Convert.ToUInt64( ds, geMean ),
            Convert.ToUInt64( ds, median ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of " +
            "the Convert.To<Type>( object, IFormatProvider ) methods " +
            "\ngenerates the following output. The example " +
            "displays the values \nreturned by the methods, " +
            "using several IFormatProvider objects.\n" );
```

```
        DataSet ds1 = new DataSet(
            10.5, 22.2, 45.9, 88.7, 156.05, 297.6 );
        DisplayDataSet( ds1 );

        DataSet ds2 = new DataSet(
            359999.95, 425000, 499999.5, 775000, 1695000 );
        DisplayDataSet( ds2 );
    }
}

/*
This example of the Convert.To<Type>( object, IFormatProvider ) methods
generates the following output. The example displays the values
returned by the methods, using several IFormatProvider objects.

DataSet: [10.5, 22.2, 45.9, 88.7, 156.05, 297.6]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       103                  59                  67
ToChar                         g                   ;                   C
ToDateTime   2003-05-13 15:04:12 2003-05-13 15:03:28 2003-05-13 15:03:35
ToDecimal       103.491666666667    59.4332135445164                67.3
ToDouble        103.491666666667    59.4332135445164                67.3
ToInt16                      103                  59                  67
ToInt32                      103                  59                  67
ToInt64                      103                  59                  67
ToSByte                      103                  59                  67
ToSingle                103.4917            59.43321                67.3
ToUInt16                     103                  59                  67
ToUInt32                     103                  59                  67
ToUInt64                     103                  59                  67

DataSet: [359999.95, 425000, 499999.5, 775000, 1695000]

Convert.                 Default      Geometric Mean              Median
--------                 -------      --------------              ------
ToBoolean                   True                True                True
ToByte                       255                 255                 255
ToChar                         ?                   ?                   ?
ToDateTime   2003-05-22 07:39:08 2003-05-20 22:28:45 2003-05-19 09:55:48
ToDecimal              750999.89    631577.237188435            499999.5
ToDouble               750999.89    631577.237188435            499999.5
ToInt16                    32767               32767               32767
ToInt32                   751000              631577              500000
ToInt64                   751000              631577              500000
ToSByte                      127                 127                 127
ToSingle                750999.9            631577.3            499999.5
ToUInt16                   65535               65535               65535
ToUInt32                  751000              631577              500000
ToUInt64                  751000              631577              500000
*/
```

Remarks

The return value is the result of invoking the IConvertible.ToInt64 method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 64-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static long ToInt64 (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than Int64.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned long integers to a long integer.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
long result;
foreach (ulong number in numbers)
{
   try {
      result = Convert.ToInt64(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int64 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt64 value 0 to a Int32 value 0.
//    Converted the UInt64 value 121 to a Int32 value 121.
//    Converted the UInt64 value 340 to a Int32 value 340.
```

```
//    The UInt64 value 18446744073709551615 is outside the range of the
Int64 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static long ToInt64 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned integers to a long integer.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
long result;
foreach (uint number in numbers)
{
   result = Convert.ToInt64(number);
   Console.WriteLine("Converted the {0} value {1:N0} to the {2} value
{3:N0}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt32 value 0 to the Int64 value 0.
//    Converted the UInt32 value 121 to the Int64 value 121.
//    Converted the UInt32 value 340 to the Int64 value 340.
//    Converted the UInt32 value 4,294,967,295 to the Int64 value
4,294,967,295.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static long ToInt64 (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of 16-bit unsigned integers to a long integer.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
long result;
foreach (ushort number in numbers)
{
   try {
      result = Convert.ToInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
```

```
        Console.WriteLine("The {0} value {1} is outside the range of the
Int64 type.",
                          number.GetType().Name, number);
    }
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the Int64 value 0.
//    Converted the UInt16 value 121 to the Int64 value 121.
//    Converted the UInt16 value 340 to the Int64 value 340.
//    Converted the UInt16 value 65535 to the Int64 value 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(String)

Converts the specified string representation of a number to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (string? value);
```

Parameters

`value` String

A string that contains a number to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example attempts to convert each element in an array of numeric strings to a long integer.

C#

```
string[] values = { "One", "1.34e28", "-26.87", "-18", "-6.00",
                    " 0", "137", "1601.9", Int32.MaxValue.ToString() };
long result;

foreach (string value in values)
{
   try {
      result = Convert.ToInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int64 type.",
value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value '{1}' is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    The String value 'One' is not in a recognizable format.
//    The String value '1.34e28' is not in a recognizable format.
//    The String value '-26.87' is not in a recognizable format.
//    Converted the String value '-18' to the Int64 value -18.
```

```
//    The String value '-6.00' is not in a recognizable format.
//    Converted the String value ' 0' to the Int64 value 0.
//    Converted the String value '137' to the Int64 value 137.
//    The String value '1601.9' is not in a recognizable format.
//    Converted the String value '2147483647' to the Int64 value
2147483647.
```

Remarks

Using the ToInt64(String) method is equivalent to passing `value` to the Int64.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Int64.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 64-bit signed integer.

```
C#
```

```
[System.CLSCompliant(false)]
public static long ToInt64 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a signed byte array to a long integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
long result;

foreach (sbyte number in numbers)
{
   result = Convert.ToInt64(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the SByte value -128 to the Int64 value -128.
//       Converted the SByte value -1 to the Int64 value -1.
//       Converted the SByte value 0 to the Int64 value 0.
//       Converted the SByte value 10 to the Int64 value 10.
//       Converted the SByte value 127 to the Int64 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Object)

Converts the value of the specified object to a 64-bit signed integer.

C#

```
public static long ToInt64 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example attempts to convert each element in an object array to a long integer.

C#

```
object[] values = { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                    "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2, 16.3e42};
long result;

foreach (object value in values)
{
   try {
      result = Convert.ToInt64(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Int64 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to an Int64 exists for the {0}
value {1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value True to the Int64 value 1.
//    Converted the Int32 value -12 to the Int64 value -12.
//    Converted the Int32 value 163 to the Int64 value 163.
//    Converted the Int32 value 935 to the Int64 value 935.
//    Converted the Char value x to the Int64 value 120.
//    No conversion to an Int64 exists for the DateTime value 5/12/2009
12:00:00 AM.
//    Converted the String value 104 to the Int64 value 104.
//    The String value 103.0 is not in a recognizable format.
```

```
//    Converted the String value -1 to the Int64 value -1.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the Int64 value 100.
//    The Double value 1.63E+43 is outside the range of the Int64 type.
```

Remarks

If `value` is not `null`, this method wraps a call to the IConvertible.ToInt64 implementation of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Int64)

Returns the specified 64-bit signed integer; no actual conversion is performed.

C#

```
public static long ToInt64 (long value);
```

Parameters

`value` Int64

A 64-bit signed integer.

Returns

Int64

`value` is returned unchanged.

Examples

The following example converts each element in an array of integers to a long integer.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
long result;
foreach (int number in numbers)
{
   result = Convert.ToInt64(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int32 value -2147483648 to the Int64 value
-2147483648.
//    Converted the Int32 value -1 to the Int64 value -1.
//    Converted the Int32 value 0 to the Int64 value 0.
//    Converted the Int32 value 121 to the Int64 value 121.
//    Converted the Int32 value 340 to the Int64 value 340.
//    Converted the Int32 value 2147483647 to the Int64 value 2147483647.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of bytes to a long integer.

C#

```
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
long result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToInt64(byteValue);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     byteValue.GetType().Name, byteValue,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the Byte value 0 to the Int64 value 0.
//       Converted the Byte value 14 to the Int64 value 14.
//       Converted the Byte value 122 to the Int64 value 122.
//       Converted the Byte value 255 to the Int64 value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Char)

Converts the value of the specified Unicode character to the equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in a Char array to a long integer.

C#

```
char[] chars = { 'a', 'z', '\u0007', '\u03FF',
                 '\u7FFF', '\uFFFE' };
long result;

foreach (char ch in chars)
{
   result = Convert.ToInt64(ch);
```

```
    Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                      ch.GetType().Name, ch,
                      result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the Char value 'a' to the Int64 value 97.
//       Converted the Char value 'z' to the Int64 value 122.
//       Converted the Char value '•' to the Int64 value 7.
//       Converted the Char value 'Ɔ' to the Int64 value 1023.
//       Converted the Char value '翿' to the Int64 value 32767.
//       Converted the Char value '￾' to the Int64 value 65534.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static long ToInt64 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Int64

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToInt64(Boolean)

Converts the specified Boolean value to the equivalent 64-bit signed integer.

```
C#
public static long ToInt64 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Int64

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example converts the Boolean values `true` and `false` to long integers.

```C#
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToInt64(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToInt64(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Double)

Converts the value of the specified double-precision floating-point number to an equivalent 64-bit signed integer.

```C#
public static long ToInt64 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Int64

`value`, rounded to the nearest 64-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int64.MaxValue or less than Int64.MinValue.

Examples

The following example attempts to convert each element in an array of Double values to a long integer.

C#

```
double[] values= { Double.MinValue, -1.38e10, -1023.299, -12.98,
                   0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
long result;

foreach (double value in values)
{
   try {
      result = Convert.ToInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int64 type.",
value);
   }
}
//    -1.7976931348623157E+308 is outside the range of the Int64 type.
//    Converted the Double value '-13800000000' to the Int64 value
-13800000000.
//    Converted the Double value '-1023.299' to the Int64 value -1023.
//    Converted the Double value '-12.98' to the Int64 value -13.
//    Converted the Double value '0' to the Int64 value 0.
//    Converted the Double value '9.113E-16' to the Int64 value 0.
//    Converted the Double value '103.919' to the Int64 value 104.
//    Converted the Double value '17834.191' to the Int64 value 17834.
//    1.7976931348623157E+308 is outside the range of the Int64 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Examples

The following example converts each element in an array of 16-bit integers to a long integer.

C#

```
short[] numbers= { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
long result;

foreach (short number in numbers)
{
   result = Convert.ToInt64(number);
   Console.WriteLine("Converted the {0} value {1} to a {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int16 value -32768 to a Int32 value -32768.
//    Converted the Int16 value -1 to a Int32 value -1.
//    Converted the Int16 value 0 to a Int32 value 0.
//    Converted the Int16 value 121 to a Int32 value 121.
//    Converted the Int16 value 340 to a Int32 value 340.
//    Converted the Int16 value 32767 to a Int32 value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Int64

A 64-bit signed integer that is equivalent to `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToInt64(Decimal)

Converts the value of the specified decimal number to an equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Int64

`value`, rounded to the nearest 64-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Exceptions

OverflowException

`value` is greater than Int64.MaxValue or less than Int64.MinValue.

Examples

The following example attempts to convert each element in an array of Decimal values to a long integer.

C#

```
decimal[] values= { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                    199.55m, 9214.16m, Decimal.MaxValue };
long result;

foreach (decimal value in values)
{
   try {
      result = Convert.ToInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the Int64 type.",
                        value);
   }
}
// The example displays the following output:
//    -79228162514264337593543950335 is outside the range of the Int64
type.
//    Converted the Decimal value '-1034.23' to the Int64 value -1034.
//    Converted the Decimal value '-12' to the Int64 value -12.
//    Converted the Decimal value '0' to the Int64 value 0.
//    Converted the Decimal value '147' to the Int64 value 147.
//    Converted the Decimal value '199.55' to the Int64 value 200.
//    Converted the Decimal value '9214.16' to the Int64 value 9214.
//    79228162514264337593543950335 is outside the range of the Int64
type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToSByte Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to an 8-bit signed integer.

Overloads

Expand table

ToSByte(SByte)	Returns the specified 8-bit signed integer; no actual conversion is performed.
ToSByte(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 8-bit signed integer.
ToSByte(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 8-bit signed integer, using the specified culture-specific formatting information.
ToSByte(Object, IFormatProvider)	Converts the value of the specified object to an 8-bit signed integer, using the specified culture-specific formatting information.
ToSByte(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit signed integer.
ToSByte(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit signed integer.
ToSByte(String)	Converts the specified string representation of a number to an equivalent 8-bit signed integer.
ToSByte(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 8-bit signed integer.
ToSByte(Object)	Converts the value of the specified object to an 8-bit signed integer.
ToSByte(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 8-bit signed integer.
ToSByte(Int32)	Converts the value of the specified 32-bit signed integer to an

	equivalent 8-bit signed integer.
ToSByte(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 8-bit signed integer.
ToSByte(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 8-bit signed integer.
ToSByte(Char)	Converts the value of the specified Unicode character to the equivalent 8-bit signed integer.
ToSByte(DateTime)	Calling this method always throws InvalidCastException.
ToSByte(Boolean)	Converts the specified Boolean value to the equivalent 8-bit signed integer.
ToSByte(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 8-bit signed integer.
ToSByte(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 8-bit signed integer.
ToSByte(Decimal)	Converts the value of the specified decimal number to an equivalent 8-bit signed integer.

ToSByte(SByte)

> Important
>
> This API is not CLS-compliant.

Returns the specified 8-bit signed integer; no actual conversion is performed.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to return.

Returns

SByte

`value` is returned unchanged.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(String, Int32)

> Important
>
> This API is not CLS-compliant.

Converts the string representation of a number in a specified base to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

SByte

An 8-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

ArgumentOutOfRangeException

`value` is Empty.

FormatException

`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException

`value`, which represents a non-base 10 signed number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example attempts to interpret the elements in a string array as the binary, octal, and hexadecimal representation of numeric values in order to convert them to unsigned bytes.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int[] baseValues = { 2, 8, 16};
      string[] values = { "FF", "81", "03", "11", "8F", "01", "1C",
"111",
                          "123", "18A" };

      // Convert to each supported base.
      foreach (int baseValue in baseValues)
      {
         Console.WriteLine("Converting strings in base {0}:", baseValue);
         foreach (string value in values)
         {
            Console.Write("   '{0,-5}  -->  ", value + "'");
            try {
               Console.WriteLine(Convert.ToSByte(value, baseValue));
            }
            catch (FormatException) {
               Console.WriteLine("Bad Format");
            }
            catch (OverflowException) {
               Console.WriteLine("Out of Range");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       Converting strings in base 2:
//          'FF'    -->  Bad Format
//          '81'    -->  Bad Format
//          '03'    -->  Bad Format
//          '11'    -->  3
//          '8F'    -->  Bad Format
//          '01'    -->  1
//          '1C'    -->  Bad Format
//          '111'   -->  7
//          '123'   -->  Bad Format
//          '18A'   -->  Bad Format
//
//       Converting strings in base 8:
//          'FF'    -->  Bad Format
```

```
//          '81'    -->  Bad Format
//          '03'    -->  3
//          '11'    -->  9
//          '8F'    -->  Bad Format
//          '01'    -->  1
//          '1C'    -->  Bad Format
//          '111'   -->  73
//          '123'   -->  83
//          '18A'   -->  Bad Format
//
//       Converting strings in base 16:
//          'FF'    -->  -1
//          '81'    -->  -127
//          '03'    -->  3
//          '11'    -->  17
//          '8F'    -->  -113
//          '01'    -->  1
//          '1C'    -->  28
//          '111'   -->  Out of Range
//          '123'   -->  Out of Range
//          '18A'   -->  Out of Range
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the negative sign is not supported for non-base 10 numeric representations, the ToSByte(String, Int32) method assumes that negative numbers use two's complement representation. In other words, the method always interprets the high-order bit of a byte (bit 7) as its sign bit. As a result, it is possible to write code in which a non-base 10 number that is out of the range of the SByte data type is converted to an SByte value without the method throwing an exception. The following example converts MaxValue to its hexadecimal string representation, and then calls the ToSByte(String, Int32) method. Instead of throwing an exception, the method displays the message, "0xff converts to -1."

C#

```
// Create a hexadecimal value out of range of the SByte type.
string value = Convert.ToString(byte.MaxValue, 16);
// Convert it back to a number.
try
{
   sbyte number = Convert.ToSByte(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
```

```
{
   Console.WriteLine("Unable to convert '0x{0}' to a signed byte.",
value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method is using the appropriate numeric representation to interpret a particular value. As the following example illustrates, you can ensure that the method handles overflows appropriately by first determining whether a value represents an unsigned or a signed type when converting it to its hexadecimal string representation. Throw an exception if the original value was an unsigned type but the conversion back to a signed byte yields a value whose sign bit is on.

C#

```
// Create a hexadecimal value out of range of the SByte type.
byte sourceNumber = byte.MaxValue;
bool isSigned =
Math.Sign(Convert.ToDouble(sourceNumber.GetType().GetField("MinValue").Ge
tValue(null))) == -1;
string value = Convert.ToString(sourceNumber, 16);
sbyte targetNumber;
try
{
   targetNumber = Convert.ToSByte(value, 16);
   if (! isSigned && ((targetNumber & 0x80) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to a signed byte.",
value);
}
// Displays the following to the console:
//    Unable to convert '0xff' to a signed byte.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 8-bit signed integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (string value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`value` is `null`.

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example converts string representations of SByte values with the `ToSByte` method, using an IFormatProvider object.

C#

```
// Example of the Convert.ToSByte( string ) and
// Convert.ToSByte( string, IFormatProvider ) methods.
using System;
using System.Globalization;

class ToSByteProviderDemo
{
    static string format = "{0,-20}{1,-20}{2}";

     // Get the exception type name; remove the namespace prefix.
    static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    static void ConvertToSByte( string numericStr,
        IFormatProvider provider )
    {
        object defaultValue;
        object providerValue;

        // Convert numericStr to SByte without a format provider.
        try
        {
            defaultValue = Convert.ToSByte( numericStr );
        }
        catch( Exception ex )
        {
            defaultValue = GetExceptionType( ex );
        }
```

```
        // Convert numericStr to SByte with a format provider.
        try
        {
            providerValue = Convert.ToSByte( numericStr, provider );
        }
        catch( Exception ex )
        {
            providerValue = GetExceptionType( ex );
        }

        Console.WriteLine( format, numericStr,
            defaultValue, providerValue );
    }

    public static void Main( )
    {
        // Create a NumberFormatInfo object and set several of its
        // properties that apply to numbers.
        NumberFormatInfo provider = new NumberFormatInfo();

        // These properties affect the conversion.
        provider.NegativeSign = "neg ";
        provider.PositiveSign = "pos ";

        // These properties do not affect the conversion.
        // The input string cannot have decimal and group separators.
        provider.NumberDecimalSeparator = ".";
        provider.NumberNegativePattern = 0;

        Console.WriteLine("This example of\n" +
            "  Convert.ToSByte( string ) and \n" +
            "  Convert.ToSByte( string, IFormatProvider ) " +
            "\ngenerates the following output. It converts " +
            "several strings to \nSByte values, using " +
            "default formatting or a NumberFormatInfo object.\n" );
        Console.WriteLine( format, "String to convert",
            "Default/exception", "Provider/exception" );
        Console.WriteLine( format, "-----------------",
            "-----------------", "-----------------" );

        // Convert strings, with and without an IFormatProvider.
        ConvertToSByte( "123", provider );
        ConvertToSByte( "+123", provider );
        ConvertToSByte( "pos 123", provider );
        ConvertToSByte( "-123", provider );
        ConvertToSByte( "neg 123", provider );
        ConvertToSByte( "123.", provider );
        ConvertToSByte( "(123)", provider );
        ConvertToSByte( "128", provider );
        ConvertToSByte( "-129", provider );
    }
}

/*
```

```
This example of
  Convert.ToSByte( string ) and
  Convert.ToSByte( string, IFormatProvider )
generates the following output. It converts several strings to
SByte values, using default formatting or a NumberFormatInfo object.

String to convert   Default/exception   Provider/exception
-----------------   -----------------   ------------------
123                 123                 123
+123                123                 FormatException
pos 123             FormatException     123
-123                -123                FormatException
neg 123             FormatException     -123
123.                FormatException     FormatException
(123)               FormatException     FormatException
128                 OverflowException   OverflowException
-129                OverflowException   FormatException
*/
```

Remarks

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the SByte.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Object, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to an 8-bit signed integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example defines a `ByteString` class that stores both signed and unsigned bytes as hexadecimal strings along with a field that indicates the sign of the byte. The `ByteString` class implements the IConvertible interface. Its IConvertible.ToSByte method calls the Parse(String, IFormatProvider) method to perform the conversion. If it fails, it throws an OverflowException.

C#

```
using System;
using System.Globalization;

public enum SignBit { Negative=-1, Zero=0, Positive=1 };

public struct ByteString : IConvertible
{
   private SignBit signBit;
   private string byteString;

   public SignBit Sign
   {
      set { signBit = value; }
      get { return signBit; }
   }

   public string Value
   {
      set {
         if (value.Trim().Length > 2)
            throw new ArgumentException("The string representation of a
byte cannot have more than two characters.");
         else
            byteString = value;
      }
      get { return byteString; }
   }
```

```
   // IConvertible implementations.
   public TypeCode GetTypeCode() {
      return TypeCode.Object;
   }

   public bool ToBoolean(IFormatProvider provider)
   {
      if (signBit == SignBit.Zero)
         return false;
      else
         return true;
   }

   public byte ToByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is out of range
of the Byte type.", Convert.ToSByte(byteString, 16)));
      else
         return Byte.Parse(byteString, NumberStyles.HexNumber);
   }

   public char ToChar(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative) {
         throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToSByte(byteString, 16)));
      }
      else {
         byte byteValue = Byte.Parse(this.byteString,
NumberStyles.HexNumber);
         return Convert.ToChar(byteValue);
      }
   }

   public DateTime ToDateTime(IFormatProvider provider)
   {
      throw new InvalidCastException("ByteString to DateTime conversion
is not supported.");
   }

   public decimal ToDecimal(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
      {
         sbyte byteValue = SByte.Parse(byteString,
NumberStyles.HexNumber);
         return Convert.ToDecimal(byteValue);
      }
      else
      {
         byte byteValue = Byte.Parse(byteString, NumberStyles.HexNumber);
         return Convert.ToDecimal(byteValue);
      }
```

```
    }

    public double ToDouble(IFormatProvider provider)
    {
       if (signBit == SignBit.Negative)
          return Convert.ToDouble(SByte.Parse(byteString,
NumberStyles.HexNumber));
       else
          return Convert.ToDouble(Byte.Parse(byteString,
NumberStyles.HexNumber));
    }

    public short ToInt16(IFormatProvider provider)
    {
       if (signBit == SignBit.Negative)
          return Convert.ToInt16(SByte.Parse(byteString,
NumberStyles.HexNumber));
       else
          return Convert.ToInt16(Byte.Parse(byteString,
NumberStyles.HexNumber));
    }

    public int ToInt32(IFormatProvider provider)
    {
       if (signBit == SignBit.Negative)
          return Convert.ToInt32(SByte.Parse(byteString,
NumberStyles.HexNumber));
       else
          return Convert.ToInt32(Byte.Parse(byteString,
NumberStyles.HexNumber));
    }

    public long ToInt64(IFormatProvider provider)
    {
       if (signBit == SignBit.Negative)
          return Convert.ToInt64(SByte.Parse(byteString,
NumberStyles.HexNumber));
       else
          return Convert.ToInt64(Byte.Parse(byteString,
NumberStyles.HexNumber));
    }

    public sbyte ToSByte(IFormatProvider provider)
    {
       try {
          return SByte.Parse(byteString, NumberStyles.HexNumber);
       }
       catch (OverflowException e) {
          throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",
                                                      Byte.Parse(byteString,
NumberStyles.HexNumber)), e);
       }
    }
```

```
   public float ToSingle(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToSingle(SByte.Parse(byteString,
NumberStyles.HexNumber));
      else
         return Convert.ToSingle(Byte.Parse(byteString,
NumberStyles.HexNumber));
   }

   public string ToString(IFormatProvider provider)
   {
      return "0x" + this.byteString;
   }

   public object ToType(Type conversionType, IFormatProvider provider)
   {
      switch (Type.GetTypeCode(conversionType))
      {
         case TypeCode.Boolean:
            return this.ToBoolean(null);
         case TypeCode.Byte:
            return this.ToByte(null);
         case TypeCode.Char:
            return this.ToChar(null);
         case TypeCode.DateTime:
            return this.ToDateTime(null);
         case TypeCode.Decimal:
            return this.ToDecimal(null);
         case TypeCode.Double:
            return this.ToDouble(null);
         case TypeCode.Int16:
            return this.ToInt16(null);
         case TypeCode.Int32:
            return this.ToInt32(null);
         case TypeCode.Int64:
            return this.ToInt64(null);
         case TypeCode.Object:
            if (typeof(ByteString).Equals(conversionType))
               return this;
            else
               throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.", conversionType.Name));
         case TypeCode.SByte:
            return this.ToSByte(null);
         case TypeCode.Single:
            return this.ToSingle(null);
         case TypeCode.String:
            return this.ToString(null);
         case TypeCode.UInt16:
            return this.ToUInt16(null);
         case TypeCode.UInt32:
            return this.ToUInt32(null);
         case TypeCode.UInt64:
            return this.ToUInt64(null);
```

```
            default:
               throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
         }
      }

      public UInt16 ToUInt16(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt16 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
         else
            return Convert.ToUInt16(Byte.Parse(byteString,
NumberStyles.HexNumber));
      }

      public UInt32 ToUInt32(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
         else
            return Convert.ToUInt32(Byte.Parse(byteString,
NumberStyles.HexNumber));
      }

      public UInt64 ToUInt64(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt64 type.",

SByte.Parse(byteString, NumberStyles.HexNumber)));
         else
            return Convert.ToUInt64(Byte.Parse(byteString,
NumberStyles.HexNumber));
      }
   }
```

The following example shows how the IConvertible.ToSByte implementation of the `ByteString` class is called by the Convert.ToSByte(Object, IFormatProvider) method.

```
C#

public class Class1
{
   public static void Main()
   {
      sbyte positiveByte = 120;
```

```
        sbyte negativeByte = -101;

        ByteString positiveString = new ByteString();
        positiveString.Sign = (SignBit) Math.Sign(positiveByte);
        positiveString.Value = positiveByte.ToString("X2");

        ByteString negativeString = new ByteString();
        negativeString.Sign = (SignBit) Math.Sign(negativeByte);
        negativeString.Value = negativeByte.ToString("X2");

        try {
           Console.WriteLine("'{0}' converts to {1}.",
positiveString.Value, Convert.ToSByte(positiveString));
        }
        catch (OverflowException) {
           Console.WriteLine("0x{0} is outside the range of the Byte
type.", positiveString.Value);
        }

        try {
           Console.WriteLine("'{0}' converts to {1}.",
negativeString.Value, Convert.ToSByte(negativeString));
        }
        catch (OverflowException) {
           Console.WriteLine("0x{0} is outside the range of the Byte
type.", negativeString.Value);
        }
    }
}
// The example displays the following output:
//       '78' converts to 120.
//       '9B' converts to -101.
```

Remarks

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of long integers to a signed byte.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
sbyte result;

foreach (ulong number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt64 value 0 to the SByte value 0.
//    Converted the UInt64 value 121 to the SByte value 121.
//    The UInt64 value 340 is outside the range of the SByte type.
//    The UInt64 value 18446744073709551615 is outside the range of the
SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToSByte(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an integer array to a signed byte.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
sbyte result;

foreach (uint number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt32 value 0 to the SByte value 0.
//    Converted the UInt32 value 121 to the SByte value 121.
//    The UInt32 value 340 is outside the range of the SByte type.
//    The UInt32 value 4294967295 is outside the range of the SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(String)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 8-bit signed integer.

```
C#

[System.CLSCompliant(false)]
public static sbyte ToSByte (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if value is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example converts string representations of SByte values with the `ToSByte` method, using default formatting.

```
C#

// Example of the Convert.ToSByte( string ) and
// Convert.ToSByte( string, IFormatProvider ) methods.
using System;
using System.Globalization;
```

```
class ToSByteProviderDemo
{
    static string format = "{0,-20}{1,-20}{2}";

     // Get the exception type name; remove the namespace prefix.
    static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    static void ConvertToSByte( string numericStr,
        IFormatProvider provider )
    {
        object defaultValue;
        object providerValue;

        // Convert numericStr to SByte without a format provider.
        try
        {
            defaultValue = Convert.ToSByte( numericStr );
        }
        catch( Exception ex )
        {
            defaultValue = GetExceptionType( ex );
        }

        // Convert numericStr to SByte with a format provider.
        try
        {
            providerValue = Convert.ToSByte( numericStr, provider );
        }
        catch( Exception ex )
        {
            providerValue = GetExceptionType( ex );
        }

        Console.WriteLine( format, numericStr,
            defaultValue, providerValue );
    }

    public static void Main( )
    {
        // Create a NumberFormatInfo object and set several of its
        // properties that apply to numbers.
        NumberFormatInfo provider = new NumberFormatInfo();

        // These properties affect the conversion.
        provider.NegativeSign = "neg ";
        provider.PositiveSign = "pos ";

        // These properties do not affect the conversion.
        // The input string cannot have decimal and group separators.
```

```
        provider.NumberDecimalSeparator = ".";
        provider.NumberNegativePattern = 0;

        Console.WriteLine("This example of\n" +
            "  Convert.ToSByte( string ) and \n" +
            "  Convert.ToSByte( string, IFormatProvider ) " +
            "\ngenerates the following output. It converts " +
            "several strings to \nSByte values, using " +
            "default formatting or a NumberFormatInfo object.\n" );
        Console.WriteLine( format, "String to convert",
            "Default/exception", "Provider/exception" );
        Console.WriteLine( format, "-----------------",
            "-----------------", "------------------" );

        // Convert strings, with and without an IFormatProvider.
        ConvertToSByte( "123", provider );
        ConvertToSByte( "+123", provider );
        ConvertToSByte( "pos 123", provider );
        ConvertToSByte( "-123", provider );
        ConvertToSByte( "neg 123", provider );
        ConvertToSByte( "123.", provider );
        ConvertToSByte( "(123)", provider );
        ConvertToSByte( "128", provider );
        ConvertToSByte( "-129", provider );
    }
}

/*
This example of
  Convert.ToSByte( string ) and
  Convert.ToSByte( string, IFormatProvider )
generates the following output. It converts several strings to
SByte values, using default formatting or a NumberFormatInfo object.

String to convert   Default/exception   Provider/exception
-----------------   -----------------   ------------------
123                 123                 123
+123                123                 FormatException
pos 123             FormatException     123
-123                -123                FormatException
neg 123             FormatException     -123
123.                FormatException     FormatException
(123)               FormatException     FormatException
128                 OverflowException   OverflowException
-129                OverflowException   FormatException
*/
```

Remarks

Using the ToSByte(String) method is equivalent to passing `value` to the SByte.Parse(String) method. `value` is interpreted by using the formatting

conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the SByte.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Single)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified single-precision floating-point number to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

SByte

`value`, rounded to the nearest 8-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of Single values to a signed byte.

C#

```
object[] values = { true, -12, 163, 935, 'x', "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2};
sbyte result;

foreach (object value in values)
{
   try {
      result = Convert.ToSByte(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
```

```
    }
    catch (InvalidCastException) {
        Console.WriteLine("No conversion to a Byte exists for the {0} value
{1}.",
                          value.GetType().Name, value);
    }
}
// The example displays the following output:
//    Converted the Boolean value true to the SByte value 1.
//    Converted the Int32 value -12 to the SByte value -12.
//    The Int32 value 163 is outside the range of the SByte type.
//    The Int32 value 935 is outside the range of the SByte type.
//    Converted the Char value x to the SByte value 120.
//    Converted the String value 104 to the SByte value 104.
//    The String value 103.0 is not in a recognizable format.
//    Converted the String value -1 to the SByte value -1.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the SByte value 100.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Object)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to an 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example attempts to convert each element in an object array to a signed byte.

C#

```
object[] values = { true, -12, 163, 935, 'x', "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2};
sbyte result;

foreach (object value in values)
{
   try {
      result = Convert.ToSByte(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to a Byte exists for the {0} value
{1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value true to the SByte value 1.
//    Converted the Int32 value -12 to the SByte value -12.
//    The Int32 value 163 is outside the range of the SByte type.
//    The Int32 value 935 is outside the range of the SByte type.
//    Converted the Char value x to the SByte value 120.
//    Converted the String value 104 to the SByte value 104.
//    The String value 103.0 is not in a recognizable format.
//    Converted the String value -1 to the SByte value -1.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the SByte value 100.
```

Remarks

The return value is the result of invoking the IConvertible.ToSByte method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned 16-bit integers to a signed byte.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
sbyte result;

foreach (ushort number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the SByte value 0.
//    Converted the UInt16 value 121 to the SByte value 121.
//    The UInt16 value 340 is outside the range of the SByte type.
//    The UInt16 value 65535 is outside the range of the SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToSByte(Int32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit signed integer to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of signed integers to a signed byte.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
sbyte result;

foreach (int number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int32 value -2147483648 is outside the range of the SByte type.
//    Converted the Int32 value -1 to the SByte value -1.
//    Converted the Int32 value 0 to the SByte value 0.
//    Converted the Int32 value 121 to the SByte value 121.
//    The Int32 value 340 is outside the range of the SByte type.
//    The Int32 value 2147483647 is outside the range of the SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Int64)

Important

This API is not CLS-compliant.

Converts the value of the specified 64-bit signed integer to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (long value);
```

Parameters

value Int64

The 64-bit signed integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of long integers to a signed byte.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
sbyte result;
foreach (long number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
```

```
                        result.GetType().Name, result);
    }
    catch (OverflowException) {
       Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
    }
}
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
SByte type.
//    Converted the Int64 value -1 to the SByte value -1.
//    Converted the Int64 value 0 to the SByte value 0.
//    Converted the Int64 value 121 to the SByte value 121.
//    The Int64 value 340 is outside the range of the SByte type.
//    The Int64 value 9223372036854775807 is outside the range of the
SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Byte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit unsigned integer to the equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue.

Examples

The following example attempts to convert each element in a byte array to a signed byte.

C#

```
byte[] numbers = { Byte.MinValue, 10, 100, Byte.MaxValue };
sbyte result;

foreach (byte number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
```

```
}
// The example displays the following output:
//    Converted the Byte value 0 to the SByte value 0.
//    Converted the Byte value 10 to the SByte value 10.
//    Converted the Byte value 100 to the SByte value 100.
//    The Byte value 255 is outside the range of the SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Char)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified Unicode character to the equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue.

Examples

The following example attempts to convert each element in an array of Char values to a signed byte.

C#

```
char[] chars = { 'a', 'z', '\u0007', '\u0200', '\u1023' };
foreach (char ch in chars)
{
   try {
      sbyte result = Convert.ToSByte(ch);
      Console.WriteLine("{0} is converted to {1}.", ch, result);
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to convert u+{0} to a byte.",
                        Convert.ToInt16(ch).ToString("X4"));
   }
}
// The example displays the following output:
//    a is converted to 97.
//    z is converted to 122.
//     is converted to 7.
//    Unable to convert u+00C8 to a byte.
//    Unable to convert u+03FF to a byte.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(DateTime)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

SByte

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToSByte(Boolean)

> Important
>
> This API is not CLS-compliant.

Converts the specified Boolean value to the equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

SByte

The number 1 if `value` is `true`; otherwise, 0.

Attributes CLSCompliantAttribute

Examples

The following example converts the Boolean values `true` and `false` to signed byte values.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToSByte(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToSByte(trueFlag));
// The example displays the following output:
//       false converts to 0.
//       true converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Double)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified double-precision floating-point number to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
```

```
public static sbyte ToSByte (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

SByte

`value`, rounded to the nearest 8-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of Double values to a signed byte.

C#

```
double[] numbers = { Double.MinValue, -129.5, -12.7, 0, 16,
                     103.6, 255.0, 1.63509e17, Double.MaxValue};
sbyte result;

foreach (double number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
```

```
    }
}
// The example displays the following output:
//    The Double value -1.79769313486232E+308 is outside the range of the
SByte type.
//    The Double value -129.5 is outside the range of the SByte type.
//    Converted the Double value -12.7 to the SByte value -13.
//    Converted the Double value 0 to the SByte value 0.
//    Converted the Double value 16 to the SByte value 16.
//    Converted the Double value 103.6 to the SByte value 104.
//    The Double value 255 is outside the range of the SByte type.
//    The Double value 1.63509E+17 is outside the range of the SByte
type.
//    The Double value 1.79769313486232E+308 is outside the range of the
SByte type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Int16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit signed integer to the equivalent 8-bit signed integer.

```C#
[System.CLSCompliant(false)]
public static sbyte ToSByte (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

SByte

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of signed 16-bit integers to a signed byte.

```C#
short[] numbers = { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
sbyte result;
foreach (short number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
```

```
}
// The example displays the following output:
//    The Int16 value -32768 is outside the range of the SByte type.
//    Converted the Int16 value -1 to the SByte value -1.
//    Converted the Int16 value 0 to the SByte value 0.
//    Converted the Int16 value 121 to the SByte value 121.
//    The Int16 value 340 is outside the range of the SByte type.
//    The Int16 value 32767 is outside the range of the SByte type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSByte(Decimal)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified decimal number to an equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

SByte

`value`, rounded to the nearest 8-bit signed integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than SByte.MaxValue or less than SByte.MinValue.

Examples

The following example attempts to convert each element in an array of Decimal values to a signed byte.

C#

```
decimal[] numbers = { Decimal.MinValue, -129.5m, -12.7m, 0m, 16m,
                      103.6m, 255.0m, Decimal.MaxValue };
sbyte result;

foreach (decimal number in numbers)
{
   try {
      result = Convert.ToSByte(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
SByte type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Decimal value -79228162514264337593543950335 is outside the
range of the SByte type.
//    The Decimal value -129.5 is outside the range of the SByte type.
//    Converted the Decimal value -12.7 to the SByte value -13.
//    Converted the Decimal value 0 to the SByte value 0.
//    Converted the Decimal value 16 to the SByte value 16.
//    Converted the Decimal value 103.6 to the SByte value 104.
```

```
//    The Decimal value 255 is outside the range of the SByte type.
//    The Decimal value 79228162514264337593543950335 is outside the
range of the SByte type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToSingle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a single-precision floating-point number.

Overloads

Expand table

ToSingle(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent single-precision floating-point number, using the specified culture-specific formatting information.
ToSingle(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent single-precision floating-point number.
ToSingle(Object, IFormatProvider)	Converts the value of the specified object to an single-precision floating-point number, using the specified culture-specific formatting information.
ToSingle(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent single-precision floating-point number.
ToSingle(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent single-precision floating-point number.
ToSingle(String)	Converts the specified string representation of a number to an equivalent single-precision floating-point number.
ToSingle(Single)	Returns the specified single-precision floating-point number; no actual conversion is performed.
ToSingle(Object)	Converts the value of the specified object to a single-precision floating-point number.
ToSingle(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent single-precision floating-point number.
ToSingle(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent single-precision floating-point number.

ToSingle(Int16)	Converts the value of the specified 16-bit signed integer to an equivalent single-precision floating-point number.
ToSingle(Double)	Converts the value of the specified double-precision floating-point number to an equivalent single-precision floating-point number.
ToSingle(Decimal)	Converts the value of the specified decimal number to an equivalent single-precision floating-point number.
ToSingle(DateTime)	Calling this method always throws InvalidCastException.
ToSingle(Char)	Calling this method always throws InvalidCastException.
ToSingle(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent single-precision floating-point number.
ToSingle(Boolean)	Converts the specified Boolean value to the equivalent single-precision floating-point number.
ToSingle(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent single-precision floating-point number.

ToSingle(String, IFormatProvider)

Converts the specified string representation of a number to an equivalent single-precision floating-point number, using the specified culture-specific formatting information.

C#

```
public static float ToSingle (string? value, IFormatProvider? provider);
```

Parameters

value String

A string that contains the number to convert.

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Single

A single-precision floating-point number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example uses IFormatProvider objects that represent the en-US and fr-FR cultures when it converts the elements in an array of numeric strings to Single values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] values = { "123456789", "12345.6789", "12 345,6789",
                          "123,456.789", "123 456,789",
"123,456,789.0123",
                          "123 456 789,0123", "1.235e12", "1.03221e-05",
                          Double.MaxValue.ToString() };
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("fr-FR") };

      foreach (CultureInfo culture in cultures)
      {
         Console.WriteLine("String -> Single Conversion Using the {0}
Culture",
                           culture.Name);
         foreach (string value in values)
         {
            Console.Write("{0,22}  ->  ", value);
            try {
               Console.WriteLine(Convert.ToSingle(value, culture));
            }
            catch (FormatException) {
               Console.WriteLine("FormatException");
```

```
            }
            catch (OverflowException) {
               Console.WriteLine("OverflowException");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    String -> Single Conversion Using the en-US Culture
//                 123456789  ->  1.234568E+08
//                12345.6789  ->  12345.68
//               12 345,6789  ->  FormatException
//               123,456.789  ->  123456.8
//               123 456,789  ->  FormatException
//          123,456,789.0123  ->  1.234568E+08
//          123 456 789,0123  ->  FormatException
//                  1.235e12  ->  1.235E+12
//               1.03221e-05  ->  1.03221E-05
//     1.79769313486232E+308  ->  Overflow
//
//    String -> Single Conversion Using the fr-FR Culture
//                 123456789  ->  1.234568E+08
//                12345.6789  ->  FormatException
//               12 345,6789  ->  12345.68
//               123,456.789  ->  FormatException
//               123 456,789  ->  123456.8
//          123,456,789.0123  ->  FormatException
//          123 456 789,0123  ->  1.234568E+08
//                  1.235e12  ->  FormatException
//               1.03221e-05  ->  FormatException
//     1.79769313486232E+308  ->  FormatException
```

Remarks

The return value is the result of invoking the Single.Parse method on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the Single.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent single-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static float ToSingle (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Single

An 8-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a signed byte array to a Single value.

C#

```
sbyte[] numbers = { SByte.MinValue, -23, 0, 17, SByte.MaxValue };
float result;

foreach (sbyte number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the SByte value '-128' to the Single value -128.
//    Converted the SByte value '-23' to the Single value -23.
//    Converted the SByte value '0' to the Single value 0.
//    Converted the SByte value '17' to the Single value 17.
//    Converted the SByte value '127' to the Single value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Object, IFormatProvider)

Converts the value of the specified object to an single-precision floating-point number, using the specified culture-specific formatting information.

C#

```
public static float ToSingle (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Single

A single-precision floating-point number that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement IConvertible.

OverflowException

`value` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example defines a `Temperature` class that implements the IConvertible interface. Its implementation of the IConvertible.ToSingle method returns the internal value of a private Single variable that represents the temperature.

C#

```
using System;
using System.Globalization;

public class Temperature : IConvertible
{
```

```
    private float m_Temp;

    public Temperature(float temperature)
    {
       this.m_Temp = temperature;
    }

    public float Celsius
    {
       get { return this.m_Temp; }
    }

    public float Kelvin
    {
       get { return this.m_Temp + 273.15f; }
    }

    public float Fahrenheit
    {
       get { return (float) Math.Round(this.m_Temp * 9 / 5 + 32, 2); }
    }

    public override string ToString()
    {
       return m_Temp.ToString("N2") + " °C";
    }

    // IConvertible implementations.
    public TypeCode GetTypeCode()
    {
       return TypeCode.Object;
    }

    public bool ToBoolean(IFormatProvider provider)
    {
       if (m_Temp == 0)
          return false;
       else
          return true;
    }

    public byte ToByte(IFormatProvider provider)
    {
       if (m_Temp < Byte.MinValue || m_Temp > Byte.MaxValue)
          throw new OverflowException(String.Format("{0} is out of range
of the Byte type.",
                                                     this.m_Temp));
       else
          return Convert.ToByte(this.m_Temp);
    }

    public char ToChar(IFormatProvider provider)
    {
       throw new InvalidCastException("Temperature to Char conversion is
not supported.");
```

```
   }

   public DateTime ToDateTime(IFormatProvider provider)
   {
      throw new InvalidCastException("Temperature to DateTime conversion
is not supported.");
   }

   public decimal ToDecimal(IFormatProvider provider)
   {
      return Convert.ToDecimal(this.m_Temp);
   }

   public double ToDouble(IFormatProvider provider)
   {
      return Convert.ToDouble(this.m_Temp);
   }

   public short ToInt16(IFormatProvider provider)
   {
      if (this.m_Temp < Int16.MinValue || this.m_Temp > Int16.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the Int16 type.",
                                                    this.m_Temp));
      else
         return Convert.ToInt16(this.m_Temp);
   }

   public int ToInt32(IFormatProvider provider)
      {
      if (this.m_Temp < Int32.MinValue || this.m_Temp > Int32.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the Int32 type.",
                                                    this.m_Temp));
      else
         return Convert.ToInt32(this.m_Temp);
   }

   public long ToInt64(IFormatProvider provider)
   {
      if (this.m_Temp < Int64.MinValue || this.m_Temp > Int64.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the Int64 type.",
                                                    this.m_Temp));
      else
         return Convert.ToInt64(this.m_Temp);
   }

   public sbyte ToSByte(IFormatProvider provider)
   {
      if (this.m_Temp < SByte.MinValue || this.m_Temp > SByte.MaxValue)
         throw new OverflowException(String.Format("{0} is out of range
of the SByte type.",
                                                    this.m_Temp));
      else
```

```
            return Convert.ToSByte(this.m_Temp);
        }

        public float ToSingle(IFormatProvider provider)
        {
            return this.m_Temp;
        }

        public string ToString(IFormatProvider provider)
        {
            return m_Temp.ToString("N2", provider) + " °C";
        }

        public object ToType(Type conversionType, IFormatProvider provider)
        {
            switch (Type.GetTypeCode(conversionType))
            {
                case TypeCode.Boolean:
                    return this.ToBoolean(null);
                case TypeCode.Byte:
                    return this.ToByte(null);
                case TypeCode.Char:
                    return this.ToChar(null);
                case TypeCode.DateTime:
                    return this.ToDateTime(null);
                case TypeCode.Decimal:
                    return this.ToDecimal(null);
                case TypeCode.Double:
                    return this.ToDouble(null);
                case TypeCode.Int16:
                    return this.ToInt16(null);
                case TypeCode.Int32:
                    return this.ToInt32(null);
                case TypeCode.Int64:
                    return this.ToInt64(null);
                case TypeCode.Object:
                    if (typeof(Temperature).Equals(conversionType))
                        return this;
                    else
                        throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.",

conversionType.Name));
                case TypeCode.SByte:
                    return this.ToSByte(null);
                case TypeCode.Single:
                    return this.ToSingle(null);
                case TypeCode.String:
                    return this.ToString(provider);
                case TypeCode.UInt16:
                    return this.ToUInt16(null);
                case TypeCode.UInt32:
                    return this.ToUInt32(null);
                case TypeCode.UInt64:
                    return this.ToUInt64(null);
```

```
            default:
               throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
         }
      }

      public ushort ToUInt16(IFormatProvider provider)
      {
         if (this.m_Temp < UInt16.MinValue || this.m_Temp > UInt16.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the UInt16 type.",
                                                       this.m_Temp));
         else
            return Convert.ToUInt16(this.m_Temp);
      }

      public uint ToUInt32(IFormatProvider provider)
      {
         if (this.m_Temp < UInt32.MinValue || this.m_Temp > UInt32.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the UInt32 type.",
                                                       this.m_Temp));
         else
            return Convert.ToUInt32(this.m_Temp);
      }

      public ulong ToUInt64(IFormatProvider provider)
      {
         if (this.m_Temp < UInt64.MinValue || this.m_Temp > UInt64.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range
of the UInt64 type.",
                                                       this.m_Temp));
         else
            return Convert.ToUInt64(this.m_Temp);
      }
   }
```

The following example illustrates how a call to the Convert.ToSingle(Object, IFormatProvider) method, in turn, calls the IConvertible.ToSingle implementation of the `Temperature` class.

C#

```
public class Example
{
   public static void Main()
   {
      Temperature cold = new Temperature(-40);
      Temperature freezing = new Temperature(0);
      Temperature boiling = new Temperature(100);

      Console.WriteLine(Convert.ToInt32(cold, null));
```

```
        Console.WriteLine(Convert.ToInt32(freezing, null));
        Console.WriteLine(Convert.ToDouble(boiling, null));
    }
}
// The example dosplays the following output:
//       -40
//       0
//       100
```

Remarks

The return value is the result of invoking the IConvertible.ToSingle method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent single-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static float ToSingle (ulong value);
```

Parameters

value UInt64

The 64-bit unsigned integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned long integers to a Single value.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 12345, UInt64.MaxValue };
float result;

foreach (ulong number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt64 value '0' to the Single value 0.
//    Converted the UInt64 value '121' to the Single value 121.
//    Converted the UInt64 value '12345' to the Single value 12345.
//    Converted the UInt64 value '18446744073709551615' to the Single
value 1.844674E+19.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent single-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static float ToSingle (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned integers to a Single value.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 12345, UInt32.MaxValue };
float result;

foreach (uint number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt32 value '0' to the Single value 0.
//    Converted the UInt32 value '121' to the Single value 121.
//    Converted the UInt32 value '12345' to the Single value 12345.
//    Converted the UInt32 value '4294967295' to the Single value
4.294967E+09.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(String)

Converts the specified string representation of a number to an equivalent single-precision floating-point number.

C#

```
public static float ToSingle (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

Single

A single-precision floating-point number that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Exceptions

FormatException

`value` is not a number in a valid format.

OverflowException

`value` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example attempts to convert each element in an array of numeric strings to a Single value.

C#

```
string[] values= { "-1,035.77219", "1AFF", "1e-35", "1.63f",
                   "1,635,592,999,999,999,999,999,999", "-17.455",
                   "190.34001", "1.29e325"};
float result;

foreach (string value in values)
{
   try {
      result = Convert.ToSingle(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
```

```
    catch (FormatException) {
       Console.WriteLine("Unable to convert '{0}' to a Single.", value);
    }
    catch (OverflowException) {
       Console.WriteLine("'{0}' is outside the range of a Single.",
value);
    }
}
// The example displays the following output:
//    Converted the String value '-1,035.77219' to the Single value
-1035.772.
//    Unable to convert '1AFF' to a Single.
//    Converted the String value '1e-35' to the Single value 1E-35.
//    Unable to convert '1.63f' to a Single.
//    Converted the String value '1,635,592,999,999,999,999,999,999' to
the Single value 1.635593E+24.
//    Converted the String value '-17.455' to the Single value -17.455.
//    Converted the String value '190.34001' to the Single value 190.34.
//    1.29e325' is outside the range of a Single.
```

Remarks

Using the ToSingle(String) method is equivalent to passing `value` to the Single.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the Single.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Single)

Returns the specified single-precision floating-point number; no actual conversion is performed.

```
C#
public static float ToSingle (float value);
```

Parameters

`value` Single

The single-precision floating-point number to return.

Returns

Single

`value` is returned unchanged.

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Object)

Converts the value of the specified object to a single-precision floating-point number.

```
C#

public static float ToSingle (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

Single

A single-precision floating-point number that is equivalent to `value`, or zero if `value` is `null`.

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example attempts to convert each element in an object array to a Single value.

```
C#
```

```
object[] values = { true, 'a', 123, 1.764e32, "9.78", "1e-02",
                    1.67e03, "A100", "1,033.67", DateTime.Now,
                    Decimal.MaxValue };
float result;

foreach (object value in values)
{
   try {
      result = Convert.ToSingle(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not recognized as a valid
Single value.",
                        value.GetType().Name, value);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
Single type.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("Conversion of the {0} value {1} to a Single is
not supported.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value 'True' to the Single value 1.
//    Conversion of the Char value a to a Single is not supported.
//    Converted the Int32 value '123' to the Single value 123.
//    Converted the Double value '1.764E+32' to the Single value
1.764E+32.
//    Converted the String value '9.78' to the Single value 9.78.
//    Converted the String value '1e-02' to the Single value 0.01.
//    Converted the Double value '1670' to the Single value 1670.
//    The String value A100 is not recognized as a valid Single value.
//    Converted the String value '1,033.67' to the Single value 1033.67.
//    Conversion of the DateTime value 11/7/2008 08:02:35 AM to a Single
is not supported.
//    Converted the Decimal value '79228162514264337593543950335' to the
Single value 7.922816E+28.
```

Remarks

The return value is the result of invoking the IConvertible.ToSingle method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent single-precision floating-point number.

C#

```
[System.CLSCompliant(false)]
public static float ToSingle (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned 16-bit integers to a Single value.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 12345, UInt16.MaxValue };
float result;

foreach (ushort number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt16 value '0' to the Single value 0.
//    Converted the UInt16 value '121' to the Single value 121.
//    Converted the UInt16 value '12345' to the Single value 12345.
//    Converted the UInt16 value '65535' to the Single value 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Int32)

Converts the value of the specified 32-bit signed integer to an equivalent single-precision floating-point number.

C#

```
public static float ToSingle (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Examples

The following example converts each element in an integer array to a Single value.

C#

```
int[] numbers = { Int32.MinValue, -1000, 0, 1000, Int32.MaxValue };
float result;

foreach (int number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int32 value '-2147483648' to the Single value
-2.147484E+09.
//    Converted the Int32 value '-1000' to the Single value -1000.
//    Converted the Int32 value '0' to the Single value 0.
//    Converted the Int32 value '1000' to the Single value 1000.
//    Converted the Int32 value '2147483647' to the Single value
2.147484E+09.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Int16)

Converts the value of the specified 16-bit signed integer to an equivalent single-precision floating-point number.

```
C#

public static float ToSingle (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Examples

The following example converts each element in an array of 16-bit integers to a Single value.

```
C#

short[] numbers = { Int16.MinValue, -1032, 0, 192, Int16.MaxValue };
float result;

foreach (short number in numbers)
{
```

```
    result = Convert.ToSingle(number);
    Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                      number.GetType().Name, number,
                      result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int16 value '-32768' to the Single value -32768.
//    Converted the Int16 value '-1032' to the Single value -1032.
//    Converted the Int16 value '0' to the Single value 0.
//    Converted the Int16 value '192' to the Single value 192.
//    Converted the Int16 value '32767' to the Single value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Double)

Converts the value of the specified double-precision floating-point number to an equivalent single-precision floating-point number.

C#

```
public static float ToSingle (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

`value` is rounded using rounding to nearest. For example, when rounded to two decimals, the value 2.345 becomes 2.34 and the value 2.355 becomes 2.36.

Examples

The following example converts each element in an array of Double values to a Single value.

C#

```
double[] values = { Double.MinValue, -1.38e10, -1023.299, -12.98,
                    0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
float result;

foreach (double value in values)
{
   result = Convert.ToSingle(value);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     value.GetType().Name, value,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Double value '-1.79769313486232E+308' to the Single
value -Infinity.
//    Converted the Double value '-13800000000' to the Single value
-1.38E+10.
//    Converted the Double value '-1023.299' to the Single value
-1023.299.
//    Converted the Double value '-12.98' to the Single value -12.98.
//    Converted the Double value '0' to the Single value 0.
//   Converted the Double value '9.113E-16' to the Single value 9.113E-
16.
//    Converted the Double value '103.919' to the Single value 103.919.
//    Converted the Double value '17834.191' to the Single value
17834.19.
//    Converted the Double value '1.79769313486232E+308' to the Single
value Infinity.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Decimal)

Converts the value of the specified decimal number to an equivalent single-precision floating-point number.

```
C#

public static float ToSingle (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

`value` is rounded using rounding to nearest. For example, when rounded to two decimals, the value 2.345 becomes 2.34 and the value 2.355 becomes 2.36.

Examples

The following example converts each element in an array of Decimal values to a Single value.

C#

```
decimal[] values = { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                             199.55m, 9214.16m, Decimal.MaxValue };
float result;

foreach (var value in values)
{
   result = Convert.ToSingle(value);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     value.GetType().Name, value,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Decimal value '-79228162514264337593543950335' to the
Single value -7.9228163E+28.
//    Converted the Decimal value '-1034.23' to the Single value
-1034.23.
//    Converted the Decimal value '-12' to the Single value -12.
//    Converted the Decimal value '0' to the Single value 0.
//    Converted the Decimal value '147' to the Single value 147.
//    Converted the Decimal value '199.55' to the Single value 199.55.
//    Converted the Decimal value '9214.16' to the Single value 9214.16.
//    Converted the Decimal value '79228162514264337593543950335' to the
Single value 7.9228163E+28.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(DateTime)

Calling this method always throws InvalidCastException.

C#

```
public static float ToSingle (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

Single

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToSingle(Char)

Calling this method always throws InvalidCastException.

C#

```
public static float ToSingle (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Single

This conversion is not supported. No value is returned.

Exceptions

InvalidCastException

This conversion is not supported.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToSingle(Byte)

Converts the value of the specified 8-bit unsigned integer to the equivalent single-precision floating-point number.

```
C#
```

```
public static float ToSingle (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Examples

The following example converts each element in an array of byte values to a Single value.

C#

```
byte[] numbers = { Byte.MinValue, 10, 100, Byte.MaxValue };
float result;

foreach (byte number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the Byte value 0 to the Single value 0.
//       Converted the Byte value 10 to the Single value 10.
//       Converted the Byte value 100 to the Single value 100.
//       Converted the Byte value 255 to the Single value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Boolean)

Converts the specified Boolean value to the equivalent single-precision floating-point number.

C#

```
public static float ToSingle (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

Single

The number 1 if `value` is `true`; otherwise, 0.

Examples

The following example converts the Boolean values `true` and `false` to Single values.

C#

```
bool[] flags = { true, false };
float result;

foreach (bool flag in flags)
{
   result = Convert.ToSingle(flag);
   Console.WriteLine("Converted {0} to {1}.", flag, result);
}
// The example displays the following output:
```

```
//       Converted True to 1.
//       Converted False to 0.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToSingle(Int64)

Converts the value of the specified 64-bit signed integer to an equivalent single-precision floating-point number.

C#

```
public static float ToSingle (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Single

A single-precision floating-point number that is equivalent to `value`.

Examples

The following example converts each element in an array of long integers to a Single value.

C#

```
long[] numbers = { Int64.MinValue, -903, 0, 172, Int64.MaxValue};
float result;

foreach (long number in numbers)
{
   result = Convert.ToSingle(number);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int64 value '-9223372036854775808' to the Single
value -9.223372E+18.
//    Converted the Int64 value '-903' to the Single value -903.
//    Converted the Int64 value '0' to the Single value 0.
//    Converted the Int64 value '172' to the Single value 172.
//    Converted the Int64 value '9223372036854775807' to the Single value
9.223372E+18.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Convert.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified value to its equivalent string representation.

Overloads

Expand table

ToString(Int32, IFormatProvider)	Converts the value of the specified 32-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int16, Int32)	Converts the value of a 16-bit signed integer to its equivalent string representation in a specified base.
ToString(DateTime, IFormatProvider)	Converts the value of the specified DateTime to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Double, IFormatProvider)	Converts the value of the specified double-precision floating-point number to its equivalent string representation.
ToString(Decimal, IFormatProvider)	Converts the value of the specified decimal number to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int32, Int32)	Converts the value of a 32-bit signed integer to its equivalent string representation in a specified base.
ToString(Int16, IFormatProvider)	Converts the value of the specified 16-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int64, IFormatProvider)	Converts the value of the specified 64-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Single, IFormatProvider)	Converts the value of the specified single-precision floating-point number to its equivalent string representation, using the specified

	culture-specific formatting information.
ToString(Object, IFormatProvider)	Converts the value of the specified object to its equivalent string representation using the specified culture-specific formatting information.
ToString(SByte, IFormatProvider)	Converts the value of the specified 8-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(String, IFormatProvider)	Returns the specified string instance; no actual conversion is performed.
ToString(UInt16, IFormatProvider)	Converts the value of the specified 16-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(UInt32, IFormatProvider)	Converts the value of the specified 32-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(UInt64, IFormatProvider)	Converts the value of the specified 64-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Char, IFormatProvider)	Converts the value of the specified Unicode character to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Int64, Int32)	Converts the value of a 64-bit signed integer to its equivalent string representation in a specified base.
ToString(Byte, Int32)	Converts the value of an 8-bit unsigned integer to its equivalent string representation in a specified base.
ToString(Boolean, IFormatProvider)	Converts the specified Boolean value to its equivalent string representation.
ToString(Byte, IFormatProvider)	Converts the value of the specified 8-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.
ToString(Boolean)	Converts the specified Boolean value to its equivalent string representation.
ToString(Byte)	Converts the value of the specified 8-bit unsigned integer to its equivalent string representation.
ToString(Char)	Converts the value of the specified Unicode character to its equivalent string representation.

ToString(DateTime)	Converts the value of the specified DateTime to its equivalent string representation.
ToString(Double)	Converts the value of the specified double-precision floating-point number to its equivalent string representation.
ToString(Int16)	Converts the value of the specified 16-bit signed integer to its equivalent string representation.
ToString(Int32)	Converts the value of the specified 32-bit signed integer to its equivalent string representation.
ToString(Decimal)	Converts the value of the specified decimal number to its equivalent string representation.
ToString(Object)	Converts the value of the specified object to its equivalent string representation.
ToString(SByte)	Converts the value of the specified 8-bit signed integer to its equivalent string representation.
ToString(Single)	Converts the value of the specified single-precision floating-point number to its equivalent string representation.
ToString(String)	Returns the specified string instance; no actual conversion is performed.
ToString(UInt16)	Converts the value of the specified 16-bit unsigned integer to its equivalent string representation.
ToString(UInt32)	Converts the value of the specified 32-bit unsigned integer to its equivalent string representation.
ToString(UInt64)	Converts the value of the specified 64-bit unsigned integer to its equivalent string representation.
ToString(Int64)	Converts the value of the specified 64-bit signed integer to its equivalent string representation.

ToString(Int32, IFormatProvider)

Converts the value of the specified 32-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (int value, IFormatProvider? provider);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(Int32, IFormatProvider) method to convert each element in an array of integers to its equivalent string representation. The conversion uses the invariant culture as well as the custom NumberFormatInfo object.

C#

```
int[] numbers = { Int32.MinValue, Int32.MaxValue};
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

foreach (int number in numbers)
   Console.WriteLine("{0,-12}  -->  {1,12}",
                     Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                     Convert.ToString(number, nfi));
// The example displays the following output:
//       -2147483648  -->  ~2147483648
//       2147483647  -->  2147483647
```

Remarks

This implementation is identical to Int32.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int16, Int32)

Converts the value of a 16-bit signed integer to its equivalent string representation in a specified base.

C#

```
public static string ToString (short value, int toBase);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

`toBase` Int32

The base of the return value, which must be 2, 8, 10, or 16.

Returns

String

The string representation of `value` in base `toBase`.

Exceptions

ArgumentException

`toBase` is not 2, 8, 10, or 16.

Examples

The following example converts each element in an array of 16-bit signed integers to its equivalent binary, octal, decimal, and hexadecimal string representations.

C#

```
int[] bases = { 2, 8, 10, 16};
short[] numbers = { Int16.MinValue, -13621, -18, 12, 19142,
Int16.MaxValue };

foreach (int baseValue in bases)
{
   Console.WriteLine("Base {0} conversion:", baseValue);
   foreach (short number in numbers)
   {
      Console.WriteLine("   {0,-8}  -->  0x{1}",
                        number, Convert.ToString(number, baseValue));
   }
}
// The example displays the following output:
//       Base 2 conversion:
//          -32768    -->  0x1000000000000000
//          -13621    -->  0x1100101011001011
//          -18       -->  0x1111111111101110
//          12        -->  0x1100
//          19142     -->  0x100101011000110
//          32767     -->  0x111111111111111
//       Base 8 conversion:
//          -32768    -->  0x100000
//          -13621    -->  0x145313
//          -18       -->  0x177756
//          12        -->  0x14
//          19142     -->  0x45306
//          32767     -->  0x77777
//       Base 10 conversion:
//          -32768    -->  0x-32768
//          -13621    -->  0x-13621
//          -18       -->  0x-18
//          12        -->  0x12
//          19142     -->  0x19142
//          32767     -->  0x32767
//       Base 16 conversion:
//          -32768    -->  0x8000
//          -13621    -->  0xcacb
//          -18       -->  0xffee
//          12        -->  0xc
//          19142     -->  0x4ac6
//          32767     -->  0x7fff
```

Remarks

If `value` is positive and `toBase` is 2, 8, or 16, the returned string uses sign-and-magnitude representation. If `value` is negative and `toBase` is 2, 8, or 16, the returned string uses two's complement representation. This means that the high-order bit of the high-order byte (bit 15) is interpreted as the sign bit. If the ToString(Int16, Int32) method is called to create a string that will later be converted back to a number, a corresponding method that assumes a similar numeric representation should be called to perform the conversion. Such methods include Convert.ToInt16(String, Int32) and Int16.Parse(String, NumberStyles).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(DateTime, IFormatProvider)

Converts the value of the specified DateTime to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (DateTime value, IFormatProvider?
provider);
```

Parameters

`value` DateTime

The date and time value to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example converts a DateTime value to its equivalent string representation in eight different cultures.

C#

```
// Specify the date to be formatted using various cultures.
DateTime tDate = new DateTime(2010, 4, 15, 20, 30, 40, 333);
// Specify the cultures.
string[] cultureNames = { "en-US", "es-AR", "fr-FR", "hi-IN",
                          "ja-JP", "nl-NL", "ru-RU", "ur-PK" };

Console.WriteLine("Converting the date {0}: ",
                  Convert.ToString(tDate,

System.Globalization.CultureInfo.InvariantCulture));

foreach (string cultureName in cultureNames)
{
   System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
   string dateString = Convert.ToString(tDate, culture);
   Console.WriteLine("   {0}:  {1,-12}",
                     culture.Name, dateString);
}
// The example displays the following output:
//       Converting the date 04/15/2010 20:30:40:
//          en-US:  4/15/2010 8:30:40 PM
//          es-AR:  15/04/2010 08:30:40 p.m.
//          fr-FR:  15/04/2010 20:30:40
//          hi-IN:  15-04-2010 20:30:40
//          ja-JP:  2010/04/15 20:30:40
//          nl-NL:  15-4-2010 20:30:40
//          ru-RU:  15.04.2010 20:30:40
//          ur-PK:  15/04/2010 8:30:40 PM
```

Remarks

This implementation is identical to DateTime.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Double, IFormatProvider)

Converts the value of the specified double-precision floating-point number to its equivalent string representation.

C#

```
public static string ToString (double value, IFormatProvider? provider);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of Double values to its equivalent string representation in four different cultures.

C#

```
// Define an array of numbers to display.
double[] numbers = { -1.5345e16, -123.4321, 19092.123, 1.1734231911290e16
};
// Define the culture names used to display them.
string[] cultureNames = { "en-US", "fr-FR", "ja-JP", "ru-RU" };

foreach (double number in numbers)
{
   Console.WriteLine("{0}:", Convert.ToString(number,

System.Globalization.CultureInfo.InvariantCulture));
   foreach (string cultureName in cultureNames)
   {
      System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
      Console.WriteLine("   {0}: {1,20}",
                        culture.Name, Convert.ToString(number, culture));
   }
   Console.WriteLine();
}
// The example displays the following output:
//    -1.5345E+16:
//       en-US:          -1.5345E+16
//       fr-FR:          -1,5345E+16
//       ja-JP:          -1.5345E+16
//       ru-RU:          -1,5345E+16
//
//    -123.4321:
//       en-US:            -123.4321
//       fr-FR:            -123,4321
//       ja-JP:            -123.4321
//       ru-RU:            -123,4321
//
//    19092.123:
//       en-US:            19092.123
//       fr-FR:            19092,123
//       ja-JP:            19092.123
//       ru-RU:            19092,123
//
//    1.173423191129E+16:
//       en-US:   1.173423191129E+16
//       fr-FR:   1,173423191129E+16
//       ja-JP:   1.173423191129E+16
//       ru-RU:   1,173423191129E+16
```

Remarks

This implementation is identical to Double.ToString(IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Decimal, IFormatProvider)

Converts the value of the specified decimal number to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (decimal value, IFormatProvider? provider);
```

Parameters

`value` Decimal

The decimal number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of Decimal values to its equivalent string representation in four different cultures.

C#

```
// Define an array of numbers to display.
decimal[] numbers = { 1734231911290.16m, -17394.32921m,
                      3193.23m, 98012368321.684m };
// Define the culture names used to display them.
string[] cultureNames = { "en-US", "fr-FR", "ja-JP", "ru-RU" };

foreach (decimal number in numbers)
{
   Console.WriteLine("{0}:", Convert.ToString(number,

System.Globalization.CultureInfo.InvariantCulture));
   foreach (string cultureName in cultureNames)
   {
      System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
      Console.WriteLine("   {0}: {1,20}",
                        culture.Name, Convert.ToString(number, culture));
   }
   Console.WriteLine();
}
// The example displays the following output:
//    1734231911290.16:
//       en-US:     1734231911290.16
//       fr-FR:     1734231911290,16
//       ja-JP:     1734231911290.16
//       ru-RU:     1734231911290,16
//
//    -17394.32921:
//       en-US:         -17394.32921
//       fr-FR:         -17394,32921
//       ja-JP:         -17394.32921
//       ru-RU:         -17394,32921
//
//    3193.23:
//       en-US:              3193.23
//       fr-FR:              3193,23
//       ja-JP:              3193.23
//       ru-RU:              3193,23
//
//    98012368321.684:
//       en-US:      98012368321.684
//       fr-FR:      98012368321,684
//       ja-JP:      98012368321.684
//       ru-RU:      98012368321,684
```

Remarks

This implementation is identical to Decimal.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int32, Int32)

Converts the value of a 32-bit signed integer to its equivalent string representation in a specified base.

C#

```
public static string ToString (int value, int toBase);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

`toBase` Int32

The base of the return value, which must be 2, 8, 10, or 16.

Returns

String

The string representation of `value` in base `toBase`.

Exceptions

ArgumentException

`toBase` is not 2, 8, 10, or 16.

Examples

The following example converts each element in an integer array to its equivalent binary, hexadecimal, decimal, and hexadecimal string representations.

C#

```
int[] bases = { 2, 8, 10, 16};
int[] numbers = { Int32.MinValue, -19327543, -13621, -18, 12,
                              19142, Int32.MaxValue };

foreach (int baseValue in bases)
{
   Console.WriteLine("Base {0} conversion:", baseValue);
   foreach (int number in numbers)
   {
      Console.WriteLine("   {0,-15}  -->  0x{1}",
                        number, Convert.ToString(number, baseValue));
   }
}
// The example displays the following output:
//    Base 2 conversion:
//       -2147483648      -->  0x10000000000000000000000000000000
//       -19327543        -->  0x11111110110110010001010111001001
//       -13621           -->  0x11111111111111111100101011001011
//       -18              -->  0x11111111111111111111111111101110
//       12               -->  0x1100
//       19142            -->  0x100101011000110
//       2147483647       -->  0x11111111111111111111111111111111
//    Base 8 conversion:
//       -2147483648      -->  0x20000000000
//       -19327543        -->  0x37666212711
//       -13621           -->  0x37777745313
//       -18              -->  0x37777777756
//       12               -->  0x14
//       19142            -->  0x45306
//       2147483647       -->  0x17777777777
//    Base 10 conversion:
//       -2147483648      -->  0x-2147483648
//       -19327543        -->  0x-19327543
//       -13621           -->  0x-13621
//       -18              -->  0x-18
//       12               -->  0x12
//       19142            -->  0x19142
//       2147483647       -->  0x2147483647
//    Base 16 conversion:
//       -2147483648      -->  0x80000000
//       -19327543        -->  0xfed915c9
//       -13621           -->  0xffffcacb
//       -18              -->  0xffffffee
//       12               -->  0xc
//       19142            -->  0x4ac6
//       2147483647       -->  0x7fffffff
```

Remarks

If `value` is positive and `toBase` is 2, 8, or 16, the returned string uses sign-and-magnitude representation. If `value` is negative and `toBase` is 2, 8, or 16, the returned string uses two's complement representation. This means that the high-order bit of the highest-order byte (bit 31) is interpreted as the sign bit. If the ToString(Int32, Int32) method is called to create a string that will later be converted back to a number, a corresponding method that assumes a similar numeric representation should be called to perform the conversion. Such methods include Convert.ToInt32(String, Int32) and Int32.Parse(String, NumberStyles).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int16, IFormatProvider)

Converts the value of the specified 16-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (short value, IFormatProvider? provider);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(Int16, IFormatProvider) method to convert each element in an array of 16-bit integers to its equivalent string representation. The conversion uses the invariant culture as well as the custom NumberFormatInfo object.

C#

```
short[] numbers = { Int16.MinValue, Int16.MaxValue};
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

foreach (short number in numbers)
   Console.WriteLine("{0,-8}  -->  {1,8}",
                     Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                     Convert.ToString(number, nfi));
// The example displays the following output:
//       -32768    -->    ~32768
//       32767     -->     32767
```

Remarks

This implementation is identical to Int16.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int64, IFormatProvider)

Converts the value of the specified 64-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (long value, IFormatProvider? provider);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(Int64, IFormatProvider) method to convert each element in a long integer array to its equivalent string representation. The conversion uses the invariant culture as well as the custom NumberFormatInfo object.

C#

```
long[] numbers = { ((long) Int32.MinValue) * 2, ((long) Int32.MaxValue) *
2};
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

foreach (long number in numbers)
   Console.WriteLine("{0,-12}  -->  {1,12}",
                     Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                     Convert.ToString(number, nfi));
// The example displays the following output:
//       -4294967296  -->  ~4294967296
//       4294967294  -->  4294967294
```

Remarks

This implementation is identical to Int64.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Single, IFormatProvider)

Converts the value of the specified single-precision floating-point number to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (float value, IFormatProvider? provider);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of Single values to its equivalent string representation in four different cultures.

C#

```
// Define an array of numbers to display.
float[] numbers = { -1.5345e16f, -123.4321f, 19092.123f,
1.1734231911290e16f };
// Define the culture names used to display them.
string[] cultureNames = { "en-US", "fr-FR", "ja-JP", "ru-RU" };

foreach (float number in numbers)
{
   Console.WriteLine("{0}:", Convert.ToString(number,

System.Globalization.CultureInfo.InvariantCulture));
   foreach (string cultureName in cultureNames)
   {
      System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
      Console.WriteLine("   {0}: {1,20}",
                        culture.Name, Convert.ToString(number, culture));
   }
   Console.WriteLine();
}
// The example displays the following output:
//    -1.5345E+16:
//       en-US:          -1.5345E+16
//       fr-FR:          -1,5345E+16
```

```
//       ja-JP:          -1.5345E+16
//       ru-RU:          -1,5345E+16
//
//    -123.4321:
//       en-US:            -123.4321
//       fr-FR:            -123,4321
//       ja-JP:            -123.4321
//       ru-RU:            -123,4321
//
//    19092.123:
//       en-US:            19092.123
//       fr-FR:            19092,123
//       ja-JP:            19092.123
//       ru-RU:            19092,123
//
//    1.173423191129E+16:
//       en-US:   1.173423191129E+16
//       fr-FR:   1,173423191129E+16
//       ja-JP:   1.173423191129E+16
//       ru-RU:   1,173423191129E+16
```

Remarks

This implementation is identical to Single.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Object, IFormatProvider)

Converts the value of the specified object to its equivalent string representation using the specified culture-specific formatting information.

```
C#
```

```
public static string? ToString (object? value, IFormatProvider?
provider);
```

Parameters

`value` Object

An object that supplies the value to convert, or `null`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`, or Empty if `value` is an object whose value is `null`. If `value` is `null`, the method returns `null`.

Examples

The following example defines a `Temperature` class that overrides the Object.ToString method but does not implement the IConvertible interface. The example illustrates how calls to the Convert.ToString(Object, IFormatProvider) method, in turn, call the `Temperature.ToString` method.

C#

```
using System;

public class Temperature
{
   private decimal m_Temp;

   public Temperature(decimal temperature)
   {
      this.m_Temp = temperature;
   }

   public decimal Celsius
   {
      get { return this.m_Temp; }
   }

   public decimal Kelvin
   {
```

```
        get { return this.m_Temp + 273.15m; }
    }

    public decimal Fahrenheit
    {
        get { return Math.Round((decimal) (this.m_Temp * 9 / 5 + 32), 2); }
    }

    public override string ToString()
    {
        return m_Temp.ToString("N2") + " °C";
    }
}

public class Example
{
    public static void Main()
    {
        Temperature cold = new Temperature(-40);
        Temperature freezing = new Temperature(0);
        Temperature boiling = new Temperature(100);

        Console.WriteLine(Convert.ToString(cold, null));
        Console.WriteLine(Convert.ToString(freezing, null));
        Console.WriteLine(Convert.ToString(boiling, null));
    }
}
// The example dosplays the following output:
//       -40.00 °C
//       0.00 °C
//       100.00 °C
```

The following example defines a `Temperature` class that implements the IFormattable interface but does not implement the IConvertible interface. Its IFormattable.ToString implementation represents the `Temperature` value in Celsius, Fahrenheit, or Kelvin, depending on the format string. The example also defines a `TemperatureProvider` class that implements IFormatProvider and provides a randomly generated format string that is used by the IFormattable implementation of the `Temperature` class.

C#

```
using System;

public class Temperature : IFormattable
{
    private decimal m_Temp;

    public Temperature(decimal temperature)
    {
        this.m_Temp = temperature;
    }
```

```
   public decimal Celsius
   { get { return this.m_Temp; } }

   public decimal Kelvin
   { get { return this.m_Temp + 273.15m; } }

   public decimal Fahrenheit
   {  get { return Math.Round(this.m_Temp * 9m / 5m + 32m, 2); } }

   public override String ToString()
   {
      return ToString("G", null);
   }

   public String ToString(String fmt, IFormatProvider provider)
   {
      TemperatureProvider formatter = null;
      if (provider != null)
         formatter = provider.GetFormat(typeof(TemperatureProvider))
                                       as TemperatureProvider;

      if (String.IsNullOrWhiteSpace(fmt)) {
         if (formatter != null)
            fmt = formatter.Format;
         else
            fmt = "G";
      }

      switch (fmt.ToUpper()) {
         case "G":
         case "C":
            return m_Temp.ToString("N2") + " °C";
         case "F":
            return Fahrenheit.ToString("N2") + " °F";
         case "K":
            return Kelvin.ToString("N2") + " K";
         default:
            throw new FormatException(String.Format("'{0}' is not a valid
format specifier.", fmt));
      }
   }
}

public class TemperatureProvider : IFormatProvider
{
   private String[] fmtStrings = { "C", "G", "F", "K" };
   private Random rnd = new Random();

   public Object GetFormat(Type formatType)
   {
      return this;
   }

   public String Format
```

```
    { get { return fmtStrings[rnd.Next(0, fmtStrings.Length)]; } }
}

public class Example
{
   public static void Main()
   {
      Temperature cold = new Temperature (-40);
      Temperature freezing = new Temperature (0);
      Temperature boiling = new Temperature (100);

      TemperatureProvider tp = new TemperatureProvider();

      Console.WriteLine(Convert.ToString(cold, tp));
      Console.WriteLine(Convert.ToString(freezing, tp));
      Console.WriteLine(Convert.ToString(boiling, tp));
   }
}
// The example displays output like the following:
//       -40.00 °C
//       273.15 K
//       100.00 °C
```

Remarks

If the `value` parameter implements the IConvertible interface, the method calls the IConvertible.ToString(IFormatProvider) implementation of `value`. Otherwise, if the `value` parameter implements the IFormattable interface, the method calls its IFormattable.ToString(String, IFormatProvider) implementation. If `value` implements neither interface, the method calls the `value` parameter's `ToString()` method, and the `provider` parameter is ignored.

The `provider` parameter is used if the `value` parameter implements the IConvertible or IFormattable interface. The most common use of the `provider` parameter is to specify culture-specific information used in the conversion of `value`. For example, if the `value` parameter is a negative decimal number, the `provider` parameter can supply culture-specific information about the notation used for the negative sign and decimal separator. The second example in the next section illustrates a format provider that does not supply culture-sensitive formatting information.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(SByte, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static string ToString (sbyte value, IFormatProvider? provider);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the Convert.ToString(SByte, IFormatProvider) method to convert each element in signed byte array to its equivalent string representation. The conversion uses the invariant culture as well as the custom NumberFormatInfo object.

C#

```
sbyte[] numbers = { SByte.MinValue, -12, 17, SByte.MaxValue};
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";
foreach (sbyte number in numbers)
   Console.WriteLine(Convert.ToString(number, nfi));
// The example displays the following output:
//       ~128
//       ~12
//       17
//       127
```

Remarks

This implementation is identical to SByte.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Returns the specified string instance; no actual conversion is performed.

```
C#

public static string? ToString (string? value, IFormatProvider?
provider);
```

Parameters

`value` String

The string to return.

`provider` IFormatProvider

An object that supplies culture-specific formatting information. This parameter is ignored.

Returns

String

`value` is returned unchanged.

Examples

The following example calls the `ToString` method with a String parameter. The method returns the unmodified String without referencing the IFormatProvider object.

```
C#

// Example of Convert.ToString( non-numeric types, IFormatProvider ).
using System;
using System.Globalization;

// An instance of this class can be passed to methods that require
// an IFormatProvider.
public class DummyProvider : IFormatProvider
{
    // Normally, GetFormat returns an object of the requested type
    // (usually itself) if it is able; otherwise, it returns Nothing.
    public object GetFormat( Type argType )
    {
        // Here, the type of argType is displayed, and GetFormat
        // always returns Nothing.
        Console.Write( "{0,-40}", argType.ToString( ) );
        return null;
```

```
    }
}

class ConvertNonNumericProviderDemo
{
    static void Main( )
    {
        // Create an instance of the IFormatProvider.
        DummyProvider provider = new DummyProvider( );
        string converted;

        // Convert these values using DummyProvider.
        int      Int32A     = -252645135;
        double   DoubleA    = 61680.3855;
        object   ObjDouble  = (object)( -98765.4321 );
        DateTime DayTimeA   = new DateTime( 2001, 9, 11, 13, 45, 0 );

        bool     BoolA      = true;
        string   StringA    = "Qwerty";
        char     CharA      = '$';
        TimeSpan TSpanA     = new TimeSpan( 0, 18, 0 );
        object   ObjOther   = (object)provider;

        Console.WriteLine( "This example of " +
            "Convert.ToString( non-numeric, IFormatProvider ) \n" +
            "generates the following output. The provider type, " +
            "argument type, \nand argument value are displayed." );
        Console.WriteLine( "\nNote: The IFormatProvider object is " +
            "not called for Boolean, String, \nChar, TimeSpan, " +
            "and non-numeric Object." );

        // The format provider is called for these conversions.
        Console.WriteLine( );
        converted = Convert.ToString( Int32A, provider );
        Console.WriteLine( "int      {0}", converted );
        converted = Convert.ToString( DoubleA, provider );
        Console.WriteLine( "double   {0}", converted );
        converted = Convert.ToString( ObjDouble, provider );
        Console.WriteLine( "object   {0}", converted );
        converted = Convert.ToString( DayTimeA, provider );
        Console.WriteLine( "DateTime {0}", converted );

        // The format provider is not called for these conversions.
        Console.WriteLine( );
        converted = Convert.ToString( BoolA, provider );
        Console.WriteLine( "bool     {0}", converted );
        converted = Convert.ToString( StringA, provider );
        Console.WriteLine( "string   {0}", converted );
        converted = Convert.ToString( CharA, provider );
        Console.WriteLine( "char     {0}", converted );
        converted = Convert.ToString( TSpanA, provider );
        Console.WriteLine( "TimeSpan {0}", converted );
        converted = Convert.ToString( ObjOther, provider );
        Console.WriteLine( "object   {0}", converted );
    }
```

```
}

/*
This example of Convert.ToString( non-numeric, IFormatProvider )
generates the following output. The provider type, argument type,
and argument value are displayed.

Note: The IFormatProvider object is not called for Boolean, String,
Char, TimeSpan, and non-numeric Object.

System.Globalization.NumberFormatInfo   int      -252645135
System.Globalization.NumberFormatInfo   double   61680.3855
System.Globalization.NumberFormatInfo   object   -98765.4321
System.Globalization.DateTimeFormatInfo DateTime 9/11/2001 1:45:00 PM

bool     True
string   Qwerty
char     $
TimeSpan 00:18:00
object   DummyProvider

*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToString(UInt16, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static string ToString (ushort value, IFormatProvider? provider);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(UInt16, IFormatProvider) method to convert a 16-bit unsigned integer value to its equivalent string representation. The conversion uses both the invariant culture and the custom NumberFormatInfo object. The output indicates that this formatting information is not used, because by default the "G" format specifier does not include a positive sign with positive values.

C#

```
ushort number = UInt16.MaxValue;
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

Console.WriteLine("{0,-6}  -->  {1,6}",
                  Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                  Convert.ToString(number, nfi));
// The example displays the following output:
//       65535   -->   65535
```

Remarks

This implementation is identical to UInt16.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(UInt32, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static string ToString (uint value, IFormatProvider? provider);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(UInt32, IFormatProvider) method to convert an unsigned integer value to its equivalent string representation. The conversion uses both the invariant culture and the custom NumberFormatInfo object. The output indicates that this formatting information is not used, because by default the "G" format specifier does not include a positive sign with positive values.

C#

```
uint number = UInt32.MaxValue;
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

Console.WriteLine("{0,-8}  -->  {1,8}",
                  Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                  Convert.ToString(number, nfi));
// The example displays the following output:
//       4294967295  -->  4294967295
```

Remarks

This implementation is identical to UInt32.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(UInt64, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static string ToString (ulong value, IFormatProvider? provider);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example defines a custom NumberFormatInfo class that defines its negative sign as the string "~" and its positive sign as the string "!". It then calls the ToString(UInt64, IFormatProvider) method to convert an unsigned long integer value to its equivalent string representation. The conversion uses both the invariant culture and the custom NumberFormatInfo object. The output indicates that this formatting information is not used, because by default the "G" format specifier does not include a positive sign with positive values.

C#

```
ulong number = UInt64.MaxValue;
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
nfi.NegativeSign = "~";
nfi.PositiveSign = "!";

Console.WriteLine("{0,-12}  -->  {1,12}",
                  Convert.ToString(number,
System.Globalization.CultureInfo.InvariantCulture),
                  Convert.ToString(number, nfi));
// The example displays the following output:
//    18446744073709551615  -->  18446744073709551615
```

Remarks

This implementation is identical to UInt64.ToString(IFormatProvider).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Char, IFormatProvider)

Converts the value of the specified Unicode character to its equivalent string representation, using the specified culture-specific formatting information.

```
C#

public static string ToString (char value, IFormatProvider? provider);
```

Parameters

`value` Char

The Unicode character to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information. This parameter is ignored.

Returns

String

The string representation of `value`.

Examples

The following example converts a Char value to a String with the `ToString` method, using an IFormatProvider object that displays the type of the format provider for which it is called. The example shows that the IFormatProvider object is not referenced.

```
C#

// Example of Convert.ToString( non-numeric types, IFormatProvider ).
using System;
using System.Globalization;

// An instance of this class can be passed to methods that require
// an IFormatProvider.
public class DummyProvider : IFormatProvider
{
    // Normally, GetFormat returns an object of the requested type
    // (usually itself) if it is able; otherwise, it returns Nothing.
    public object GetFormat( Type argType )
    {
        // Here, the type of argType is displayed, and GetFormat
        // always returns Nothing.
```

```
        Console.Write( "{0,-40}", argType.ToString( ) );
        return null;
    }
}

class ConvertNonNumericProviderDemo
{
    static void Main( )
    {
        // Create an instance of the IFormatProvider.
        DummyProvider provider = new DummyProvider( );
        string converted;

        // Convert these values using DummyProvider.
        int      Int32A     = -252645135;
        double   DoubleA    = 61680.3855;
        object   ObjDouble  = (object)( -98765.4321 );
        DateTime DayTimeA   = new DateTime( 2001, 9, 11, 13, 45, 0 );

        bool     BoolA      = true;
        string   StringA    = "Qwerty";
        char     CharA      = '$';
        TimeSpan TSpanA     = new TimeSpan( 0, 18, 0 );
        object   ObjOther   = (object)provider;

        Console.WriteLine( "This example of " +
            "Convert.ToString( non-numeric, IFormatProvider ) \n" +
            "generates the following output. The provider type, " +
            "argument type, \nand argument value are displayed." );
        Console.WriteLine( "\nNote: The IFormatProvider object is " +
            "not called for Boolean, String, \nChar, TimeSpan, " +
            "and non-numeric Object." );

        // The format provider is called for these conversions.
        Console.WriteLine( );
        converted = Convert.ToString( Int32A, provider );
        Console.WriteLine( "int      {0}", converted );
        converted = Convert.ToString( DoubleA, provider );
        Console.WriteLine( "double   {0}", converted );
        converted = Convert.ToString( ObjDouble, provider );
        Console.WriteLine( "object   {0}", converted );
        converted = Convert.ToString( DayTimeA, provider );
        Console.WriteLine( "DateTime {0}", converted );

        // The format provider is not called for these conversions.
        Console.WriteLine( );
        converted = Convert.ToString( BoolA, provider );
        Console.WriteLine( "bool     {0}", converted );
        converted = Convert.ToString( StringA, provider );
        Console.WriteLine( "string   {0}", converted );
        converted = Convert.ToString( CharA, provider );
        Console.WriteLine( "char     {0}", converted );
        converted = Convert.ToString( TSpanA, provider );
        Console.WriteLine( "TimeSpan {0}", converted );
        converted = Convert.ToString( ObjOther, provider );
```

```
        Console.WriteLine( "object   {0}", converted );
    }
}

/*
This example of Convert.ToString( non-numeric, IFormatProvider )
generates the following output. The provider type, argument type,
and argument value are displayed.

Note: The IFormatProvider object is not called for Boolean, String,
Char, TimeSpan, and non-numeric Object.

System.Globalization.NumberFormatInfo   int      -252645135
System.Globalization.NumberFormatInfo   double   61680.3855
System.Globalization.NumberFormatInfo   object   -98765.4321
System.Globalization.DateTimeFormatInfo DateTime 9/11/2001 1:45:00 PM

bool     True
string   Qwerty
char     $
TimeSpan 00:18:00
object   DummyProvider

*/
```

Remarks

This implementation is identical to Char.ToString(). The `provider` parameter is ignored.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int64, Int32)

Converts the value of a 64-bit signed integer to its equivalent string representation in a specified base.

```
C#

public static string ToString (long value, int toBase);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

`toBase` Int32

The base of the return value, which must be 2, 8, 10, or 16.

Returns

String

The string representation of `value` in base `toBase`.

Exceptions

ArgumentException

`toBase` is not 2, 8, 10, or 16.

Examples

The following example converts each element in a long integer array to its equivalent binary, hexadecimal, decimal, and hexadecimal string representations.

```
C#

int[] bases = { 2, 8, 10, 16};
long[] numbers = { Int64.MinValue, -193275430, -13621, -18, 12,
                   1914206117, Int64.MaxValue };

foreach (int baseValue in bases)
{
   Console.WriteLine("Base {0} conversion:", baseValue);
   foreach (long number in numbers)
   {
      Console.WriteLine("   {0,-23}  -->  0x{1}",
                        number, Convert.ToString(number, baseValue));
   }
```

```
}
// The example displays the following output:
//    Base 2 conversion:
//       -9223372036854775808     -->
0x1000000000000000000000000000000000000000000000000000000000000000
//       -193275430               -->
0x11111111111111111111111111111111101000111010110110011101101 0
//       -13621                   -->
0x11111111111111111111111111111111111111111111111100101011001011
//       -18                      -->
0x11111111111111111111111111111111111111111111111111111111101110
//       12                       -->  0x1100
//       1914206117               -->  0x1110010000110000111011110100101
//       9223372036854775807      -->
0x111111111111111111111111111111111111111111111111111111111111111
//    Base 8 conversion:
//       -9223372036854775808     -->  0x1000000000000000000000
//       -193275430               -->  0x1777777777776436554732
//       -13621                   -->  0x1777777777777777745313
//       -18                      -->  0x1777777777777777777756
//       12                       -->  0x14
//       1914206117               -->  0x16206073645
//       9223372036854775807      -->  0x7777777777777777777
//    Base 10 conversion:
//       -9223372036854775808     -->  0x-9223372036854775808
//       -193275430               -->  0x-193275430
//       -13621                   -->  0x-13621
//       -18                      -->  0x-18
//       12                       -->  0x12
//       1914206117               -->  0x1914206117
//       9223372036854775807      -->  0x9223372036854775807
//    Base 16 conversion:
//       -9223372036854775808     -->  0x8000000000000000
//       -193275430               -->  0xffffffff47ad9da
//       -13621                   -->  0xffffffffffffcacb
//       -18                      -->  0xffffffffffffffee
//       12                       -->  0xc
//       1914206117               -->  0x721877a5
//       9223372036854775807      -->  0x7fffffffffffffff
```

Remarks

If `value` is positive and `toBase` is not 10, the returned string uses sign-and-magnitude representation. If `value` is negative and `toBase` is not 10, the returned string uses two's complement representation. This means that the high-order bit of the highest-order byte (bit 63) is interpreted as the sign bit. If the ToString(Int64, Int32) method is called to create a string that will later be converted back to a number, a corresponding method that assumes a similar numeric representation should be called to perform the conversion. Such methods include Convert.ToInt64(String, Int32) and Int64.Parse(String, NumberStyles).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Byte, Int32)

Converts the value of an 8-bit unsigned integer to its equivalent string representation in a specified base.

C#

```
public static string ToString (byte value, int toBase);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

`toBase` Int32

The base of the return value, which must be 2, 8, 10, or 16.

Returns

String

The string representation of `value` in base `toBase`.

Exceptions

ArgumentException

`toBase` is not 2, 8, 10, or 16.

Examples

The following example converts each element in a byte array to its equivalent binary, hexadecimal, decimal, and hexadecimal string representations.

C#

```
int[] bases = { 2, 8, 10, 16};
byte[] numbers = { Byte.MinValue, 12, 103, Byte.MaxValue};

foreach (int baseValue in bases)
{
   Console.WriteLine("Base {0} conversion:", baseValue);
   foreach (byte number in numbers)
   {
      Console.WriteLine("   {0,-5}  -->  0x{1}",
                        number, Convert.ToString(number, baseValue));
   }
}
// The example displays the following output:
//       Base 2 conversion:
//          0      -->  0x0
//          12     -->  0x1100
//          103    -->  0x1100111
//          255    -->  0x11111111
//       Base 8 conversion:
//          0      -->  0x0
//          12     -->  0x14
//          103    -->  0x147
//          255    -->  0x377
//       Base 10 conversion:
//          0      -->  0x0
//          12     -->  0x12
//          103    -->  0x103
//          255    -->  0x255
//       Base 16 conversion:
//          0      -->  0x0
//          12     -->  0xc
//          103    -->  0x67
//          255    -->  0xff
```

Remarks

If `toBase` does not equal 10, the string that is returned by the Convert.ToString(Byte, Int32) method represents `value` by its magnitude only. If the method is called to create a string that will later be converted back to a number, a corresponding method that assumes a magnitude-only numeric representation should be called to perform the conversion. Such methods include Convert.ToByte(String, Int32) or Byte.Parse(String, NumberStyles).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Boolean, IFormatProvider)

Converts the specified Boolean value to its equivalent string representation.

C#

```
public static string ToString (bool value, IFormatProvider? provider);
```

Parameters

`value` Boolean

The Boolean value to convert.

`provider` IFormatProvider

An instance of an object. This parameter is ignored.

Returns

String

The string representation of `value`.

Examples

The following example converts a Boolean value to a String with the `ToString` method, using an IFormatProvider object that displays the type of the format

provider for which it is called. The example shows that the IFormatProvider object is not referenced.

C#

```
// Example of Convert.ToString( non-numeric types, IFormatProvider ).
using System;
using System.Globalization;

// An instance of this class can be passed to methods that require
// an IFormatProvider.
public class DummyProvider : IFormatProvider
{
    // Normally, GetFormat returns an object of the requested type
    // (usually itself) if it is able; otherwise, it returns Nothing.
    public object GetFormat( Type argType )
    {
        // Here, the type of argType is displayed, and GetFormat
        // always returns Nothing.
        Console.Write( "{0,-40}", argType.ToString( ) );
        return null;
    }
}

class ConvertNonNumericProviderDemo
{
    static void Main( )
    {
        // Create an instance of the IFormatProvider.
        DummyProvider provider = new DummyProvider( );
        string converted;

        // Convert these values using DummyProvider.
        int      Int32A     = -252645135;
        double   DoubleA    = 61680.3855;
        object   ObjDouble  = (object)( -98765.4321 );
        DateTime DayTimeA   = new DateTime( 2001, 9, 11, 13, 45, 0 );

        bool     BoolA      = true;
        string   StringA    = "Qwerty";
        char     CharA      = '$';
        TimeSpan TSpanA     = new TimeSpan( 0, 18, 0 );
        object   ObjOther   = (object)provider;

        Console.WriteLine( "This example of " +
            "Convert.ToString( non-numeric, IFormatProvider ) \n" +
            "generates the following output. The provider type, " +
            "argument type, \nand argument value are displayed." );
        Console.WriteLine( "\nNote: The IFormatProvider object is " +
            "not called for Boolean, String, \nChar, TimeSpan, " +
            "and non-numeric Object." );

        // The format provider is called for these conversions.
        Console.WriteLine( );
```

```
        converted = Convert.ToString( Int32A, provider );
        Console.WriteLine( "int      {0}", converted );
        converted = Convert.ToString( DoubleA, provider );
        Console.WriteLine( "double   {0}", converted );
        converted = Convert.ToString( ObjDouble, provider );
        Console.WriteLine( "object   {0}", converted );
        converted = Convert.ToString( DayTimeA, provider );
        Console.WriteLine( "DateTime {0}", converted );

        // The format provider is not called for these conversions.
        Console.WriteLine( );
        converted = Convert.ToString( BoolA, provider );
        Console.WriteLine( "bool     {0}", converted );
        converted = Convert.ToString( StringA, provider );
        Console.WriteLine( "string   {0}", converted );
        converted = Convert.ToString( CharA, provider );
        Console.WriteLine( "char     {0}", converted );
        converted = Convert.ToString( TSpanA, provider );
        Console.WriteLine( "TimeSpan {0}", converted );
        converted = Convert.ToString( ObjOther, provider );
        Console.WriteLine( "object   {0}", converted );
    }
}

/*
This example of Convert.ToString( non-numeric, IFormatProvider )
generates the following output. The provider type, argument type,
and argument value are displayed.

Note: The IFormatProvider object is not called for Boolean, String,
Char, TimeSpan, and non-numeric Object.

System.Globalization.NumberFormatInfo   int      -252645135
System.Globalization.NumberFormatInfo   double   61680.3855
System.Globalization.NumberFormatInfo   object   -98765.4321
System.Globalization.DateTimeFormatInfo DateTime 9/11/2001 1:45:00 PM

bool     True
string   Qwerty
char     $
TimeSpan 00:18:00
object   DummyProvider

*/
```

Remarks

This implementation is identical to Boolean.ToString. It returns Boolean.TrueString for `true` values and Boolean.FalseString for `false` values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Byte, IFormatProvider)

Converts the value of the specified 8-bit unsigned integer to its equivalent string representation, using the specified culture-specific formatting information.

C#

```
public static string ToString (byte value, IFormatProvider? provider);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an unsigned byte array to its equivalent string representation using the formatting conventions of the en-US and fr-FR cultures. Because the "G" specifier by default outputs only decimal digits in a

byte value's string representation, the `provider` parameter does not affect the formatting of the returned string.

C#

```
byte[] numbers = { 12, 100, Byte.MaxValue };
// Define the culture names used to display them.
string[] cultureNames = { "en-US", "fr-FR" };

foreach (byte number in numbers)
{
   Console.WriteLine("{0}:", Convert.ToString(number,

System.Globalization.CultureInfo.InvariantCulture));
   foreach (string cultureName in cultureNames)
   {
      System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
      Console.WriteLine("   {0}: {1,20}",
                        culture.Name, Convert.ToString(number, culture));
   }
   Console.WriteLine();
}
// The example displays the following output:
//       12:
//          en-US:                   12
//          fr-FR:                   12
//
//       100:
//          en-US:                  100
//          fr-FR:                  100
//
//       255:
//          en-US:                  255
//          fr-FR:                  255
```

Remarks

This implementation is identical to Byte.ToString.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Boolean)

Converts the specified Boolean value to its equivalent string representation.

C#

```
public static string ToString (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

String

The string representation of `value`.

Examples

The following example illustrates the conversion of a Boolean to a String, using `ToString`. It also illustrates that the string returned by the conversion equals either Boolean.TrueString or Boolean.FalseString.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine(Convert.ToString(falseFlag));
Console.WriteLine(Convert.ToString(falseFlag).Equals(Boolean.FalseString)
);
Console.WriteLine(Convert.ToString(trueFlag));
Console.WriteLine(Convert.ToString(trueFlag).Equals(Boolean.TrueString));
```

```
// The example displays the following output:
//       False
//       True
//       True
//       True
```

Remarks

This implementation is identical to Boolean.ToString. It returns Boolean.TrueString for `true` values and Boolean.FalseString for `false` values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Byte)

Converts the value of the specified 8-bit unsigned integer to its equivalent string representation.

C#

```
public static string ToString (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts each value in a Byte array to a string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      byte[] values = { Byte.MinValue, 12, 100, 179, Byte.MaxValue } ;

      foreach (var value in values)
         Console.WriteLine("{0,3} ({1}) --> {2}", value,
                           value.GetType().Name,
                           Convert.ToString(value));
   }
}
// The example displays the following output:
//       0 (Byte) --> 0
//      12 (Byte) --> 12
//     100 (Byte) --> 100
//     179 (Byte) --> 179
//     255 (Byte) --> 255
```

Remarks

This implementation is identical to Byte.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Char)

Converts the value of the specified Unicode character to its equivalent string representation.

C#

```
public static string ToString (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts a Char to a String.

C#

```
public void ConvertStringChar(string stringVal) {
    char charVal = 'a';

    // A string must be one character long to convert to char.
    try {
        charVal = System.Convert.ToChar(stringVal);
        System.Console.WriteLine("{0} as a char is {1}",
            stringVal, charVal);
    }
    catch (System.FormatException) {
        System.Console.WriteLine(
            "The string is longer than one character.");
    }
```

```
    catch (System.ArgumentNullException) {
        System.Console.WriteLine("The string is null.");
    }

    // A char to string conversion will always succeed.
    stringVal = System.Convert.ToString(charVal);
    System.Console.WriteLine("The character as a string is {0}",
            stringVal);
}
```

Remarks

This implementation is identical to Char.ToString.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(DateTime)

Converts the value of the specified DateTime to its equivalent string representation.

C#

```
public static string ToString (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of a DateTime value to a String value.

C#

```
DateTime[] dates = { new DateTime(2009, 7, 14),
                     new DateTime(1, 1, 1, 18, 32, 0),
                     new DateTime(2009, 2, 12, 7, 16, 0) };
string result;

foreach (DateTime dateValue in dates)
{
   result = Convert.ToString(dateValue);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        dateValue.GetType().Name, dateValue,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the DateTime value 7/14/2009 12:00:00 AM to a String
value 7/14/2009 12:00:00 AM.
//    Converted the DateTime value 1/1/0001 06:32:00 PM to a String value
1/1/0001 06:32:00 PM.
//    Converted the DateTime value 2/12/2009 07:16:00 AM to a String
value 2/12/2009 07:16:00 AM.
```

Remarks

This implementation is identical to DateTime.ToString(). It uses the formatting conventions of the current culture and the "G" format specifier to convert a DateTime value to its string representation.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Double)

Converts the value of the specified double-precision floating-point number to its equivalent string representation.

C#

```
public static string ToString (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts a Double to a String.

C#

```
public void ConvertDoubleString(double doubleVal) {

    string  stringVal;

    // A conversion from Double to string cannot overflow.
    stringVal = System.Convert.ToString(doubleVal);
    System.Console.WriteLine("{0} as a string is: {1}",
        doubleVal, stringVal);
```

```
    try {
        doubleVal = System.Convert.ToDouble(stringVal);
        System.Console.WriteLine("{0} as a double is: {1}",
            stringVal, doubleVal);
    }
    catch (System.OverflowException) {
        System.Console.WriteLine(
            "Conversion from string-to-double overflowed.");
    }
    catch (System.FormatException) {
        System.Console.WriteLine(
            "The string was not formatted as a double.");
    }
    catch (System.ArgumentException) {
        System.Console.WriteLine(
            "The string pointed to null.");
    }
}
```

Remarks

This implementation is identical to Double.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int16)

Converts the value of the specified 16-bit signed integer to its equivalent string representation.

```
C#
```

```
public static string ToString (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of 16-bit integers to its equivalent string representation.

C#

```
short[] numbers = { Int16.MinValue, -138, 0, 19, Int16.MaxValue };
string result;

foreach (short number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Int16 value -32768 to the String value -32768.
//    Converted the Int16 value -138 to the String value -138.
//    Converted the Int16 value 0 to the String value 0.
//    Converted the Int16 value 19 to the String value 19.
//    Converted the Int16 value 32767 to the String value 32767.
```

Remarks

This implementation is identical to Int16.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int32)

Converts the value of the specified 32-bit signed integer to its equivalent string representation.

C#

```
public static string ToString (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

String

The string representation of `value`.

Examples

The following example compares the ToString(Int32) method with the ToString(UInt16, IFormatProvider) method. It defines a custom NumberFormatInfo object that uses the sting "minus" to represent the negative sign. It converts each element in an integer array to its equivalent string representation using default formatting (the formatting conventions of the current culture) and the custom format provider.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set its NegativeSigns
      // property to use for integer formatting.
      NumberFormatInfo provider = new NumberFormatInfo();
      provider.NegativeSign = "minus ";

      int[] values = { -20, 0, 100 };

      Console.WriteLine("{0,-8} --> {1,10} {2,10}\n", "Value",
                        CultureInfo.CurrentCulture.Name,
                        "Custom");
      foreach (int value in values)
         Console.WriteLine("{0,-8} --> {1,10} {2,10}",
                           value, Convert.ToString(value),
                           Convert.ToString(value, provider));
      // The example displays output like the following:
      //       Value    -->      en-US     Custom
      //
      //       -20      -->        -20   minus 20
      //       0        -->          0          0
      //       100      -->        100        100
   }
}
```

Remarks

This implementation is identical to Int32.ToString(). It formats `value` by using the formatting conventions of the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString(Decimal)

Converts the value of the specified decimal number to its equivalent string representation.

C#

```
public static string ToString (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts a Decimal to a String.

C#

```
public void ConvertStringDecimal(string stringVal) {
    decimal decimalVal = 0;

    try {
        decimalVal = System.Convert.ToDecimal(stringVal);
        System.Console.WriteLine(
            "The string as a decimal is {0}.", decimalVal);
    }
    catch (System.OverflowException){
        System.Console.WriteLine(
            "The conversion from string to decimal overflowed.");
    }
    catch (System.FormatException) {
        System.Console.WriteLine(
            "The string is not formatted as a decimal.");
```

```
    }
    catch (System.ArgumentNullException) {
        System.Console.WriteLine(
            "The string is null.");
    }

    // Decimal to string conversion will not overflow.
    stringVal = System.Convert.ToString(decimalVal);
    System.Console.WriteLine(
        "The decimal as a string is {0}.", stringVal);
}
```

Remarks

This implementation is identical to Decimal.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Object)

Converts the value of the specified object to its equivalent string representation.

C#

```
public static string? ToString (object? value);
```

Parameters

`value` Object

An object that supplies the value to convert, or `null`.

Returns

String

The string representation of `value`, or Empty if `value` is `null`.

Examples

The following example converts each element in an object array to its equivalent string representation.

C#

```
object[] values = { false, 12.63m, new DateTime(2009, 6, 1, 6, 32, 15),
16.09e-12,
                    'Z', 15.15322, SByte.MinValue, Int32.MaxValue };
string result;

foreach (object value in values)
{
   result = Convert.ToString(value);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Boolean value False to the String value False.
//    Converted the Decimal value 12.63 to the String value 12.63.
//    Converted the DateTime value 6/1/2009 06:32:15 AM to the String
value 6/1/2009 06:32:15 AM.
//    Converted the Double value 1.609E-11 to the String value 1.609E-11.
//    Converted the Char value Z to the String value Z.
//    Converted the Double value 15.15322 to the String value 15.15322.
//    Converted the SByte value -128 to the String value -128.
//    Converted the Int32 value 2147483647 to the String value
2147483647.
```

Remarks

To convert `value` to its string representation, the method tries to call the IConvertible.ToString implementation of `value`. If `value` does not implement the IConvertible interface, the method tries to call the IFormattable.ToString implementation of `value`. If value does not implement the IFormattable interface, the method calls the `ToString` method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to its equivalent string representation.

C#

```
[System.CLSCompliant(false)]
public static string ToString (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a signed byte array to its equivalent string representation.

C#

```
sbyte[] numbers = { SByte.MinValue, -12, 0, 16, SByte.MaxValue };
string result;

foreach (sbyte number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the SByte value -128 to the String value -128.
//    Converted the SByte value -12 to the String value -12.
//    Converted the SByte value 0 to the String value 0.
//    Converted the SByte value 16 to the String value 16.
//    Converted the SByte value 127 to the String value 127.
```

Remarks

This implementation is identical to SByte.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Single)

Converts the value of the specified single-precision floating-point number to its equivalent string representation.

```
C#

public static string ToString (float value);
```

Parameters

value Single

The single-precision floating-point number to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in an array of Single values to its equivalent string representation.

```
C#

float[] numbers = { Single.MinValue, -1011.351f, -17.45f, -3e-16f,
                    0f, 4.56e-12f, 16.0001f, 10345.1221f, Single.MaxValue
};
string result;

foreach (float number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Single value -3.402823E+38 to the String value
-3.402823E+38.
//    Converted the Single value -1011.351 to the String value -1011.351.
//    Converted the Single value -17.45 to the String value -17.45.
//    Converted the Single value -3E-16 to the String value -3E-16.
//    Converted the Single value 0 to the String value 0.
//    Converted the Single value 4.56E-12 to the String value 4.56E-12.
//    Converted the Single value 16.0001 to the String value 16.0001.
//    Converted the Single value 10345.12 to the String value 10345.12.
//    Converted the Single value 3.402823E+38 to the String value
3.402823E+38.
```

Remarks

This implementation is identical to Single.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Returns the specified string instance; no actual conversion is performed.

C#

```
public static string? ToString (string? value);
```

Parameters

`value` String

The string to return.

Returns

String

`value` is returned unchanged.

Examples

The following example passes a string to the ToString(String) method and calls the Object.ReferenceEquals method to confirm that the method returns the original

string. The example also calls the String.IsInterned method to ensure that the two strings are not identical because the original string is interned.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String article = "An";
      String noun = "apple";
      String str1 = String.Format("{0} {1}", article, noun);
      String str2 = Convert.ToString(str1);

      Console.WriteLine("str1 is interned: {0}",
                        ! (String.IsInterned(str1) == null));
      Console.WriteLine("str1 and str2 are the same reference: {0}",
                        Object.ReferenceEquals(str1, str2));
   }
}
// The example displays the following output:
//       str1 is interned: False
//       str1 and str2 are the same reference: True
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToString(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to its equivalent string representation.

C#

```
[System.CLSCompliant(false)]
public static string ToString (ushort value);
```

Parameters

value UInt16

The 16-bit unsigned integer to convert.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned 16-bit integer values to its equivalent string representation.

C#

```
ushort[] numbers = { UInt16.MinValue, 103, 1045, UInt16.MaxValue };
string result;

foreach (ushort number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the String value 0.
//    Converted the UInt16 value 103 to the String value 103.
//    Converted the UInt16 value 1045 to the String value 1045.
//    Converted the UInt16 value 65535 to the String value 65535.
```

Remarks

This implementation is identical to UInt16.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to its equivalent string representation.

C#

```
[System.CLSCompliant(false)]
public static string ToString (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an unsigned integer array to its equivalent string representation.

C#

```
uint[] numbers = { UInt32.MinValue, 103, 1045, 119543, UInt32.MaxValue };
string result;

foreach (uint number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt32 value 0 to the String value 0.
//    Converted the UInt32 value 103 to the String value 103.
//    Converted the UInt32 value 1045 to the String value 1045.
//    Converted the UInt32 value 119543 to the String value 119543.
//    Converted the UInt32 value 4294967295 to the String value
4294967295.
```

Remarks

This implementation is identical to UInt32.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to its equivalent string representation.

C#

```
[System.CLSCompliant(false)]
public static string ToString (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

String

The string representation of `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an unsigned long integer array to its equivalent string representation.

C#

```
ulong[] numbers = { UInt64.MinValue, 1031, 189045, UInt64.MaxValue };
string result;

foreach (ulong number in numbers)
{
   result = Convert.ToString(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
```

```
}
// The example displays the following output:
//    Converted the UInt64 value 0 to the String value 0.
//    Converted the UInt64 value 1031 to the String value 1031.
//    Converted the UInt64 value 189045 to the String value 189045.
//    Converted the UInt64 value 18446744073709551615 to the String value
18446744073709551615.
```

Remarks

This implementation is identical to UInt64.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int64)

Converts the value of the specified 64-bit signed integer to its equivalent string representation.

C#

```
public static string ToString (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

String

The string representation of `value`.

Examples

The following example converts each element in a long integer array to its equivalent string representation.

C#

```
// Create a NumberFormatInfo object and set several of its
// properties that control default integer formatting.
System.Globalization.NumberFormatInfo provider = new
System.Globalization.NumberFormatInfo();
provider.NegativeSign = "minus ";

long[] values = { -200, 0, 1000 };

foreach (long value in values)
   Console.WriteLine("{0,-6}  -->  {1,10}",
                     value, Convert.ToString(value, provider));
// The example displays the following output:
//       -200    -->   minus 200
//       0       -->           0
//       1000    -->        1000
```

Remarks

This implementation is identical to Int64.ToString().

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Convert.ToUInt16 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 16-bit unsigned integer.

Overloads

Expand table

ToUInt16(String)	Converts the specified string representation of a number to an equivalent 16-bit unsigned integer.
ToUInt16(UInt16)	Returns the specified 16-bit unsigned integer; no actual conversion is performed.
ToUInt16(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit unsigned integer.
ToUInt16(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 16-bit unsigned integer.
ToUInt16(Object, IFormatProvider)	Converts the value of the specified object to a 16-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt16(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 16-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt16(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 16-bit unsigned integer.
ToUInt16(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit unsigned integer.
ToUInt16(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 16-bit unsigned integer.
ToUInt16(Char)	Converts the value of the specified Unicode character to the equivalent 16-bit unsigned integer.

ToUInt16(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 16-bit unsigned integer.
ToUInt16(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 16-bit unsigned integer.
ToUInt16(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 16-bit unsigned integer.
ToUInt16(Decimal)	Converts the value of the specified decimal number to an equivalent 16-bit unsigned integer.
ToUInt16(DateTime)	Calling this method always throws InvalidCastException.
ToUInt16(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit unsigned integer.
ToUInt16(Boolean)	Converts the specified Boolean value to the equivalent 16-bit unsigned integer.
ToUInt16(Object)	Converts the value of the specified object to a 16-bit unsigned integer.
ToUInt16(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 16-bit unsigned integer.

ToUInt16(String)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example attempts to convert each element in a numeric string array to a 16-bit unsigned integer.

C#

```
string[] values = { "1603", "1,603", "one", "1.6e03", "1.2e-02",
                    "-1326", "1074122" };
ushort result;

foreach (string value in values)
{
   try {
      result = Convert.ToUInt16(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt16 type.",
value);
```

```
    }
}
// The example displays the following output:
//    Converted the String value '1603' to the UInt16 value 1603.
//    The String value 1,603 is not in a recognizable format.
//    The String value one is not in a recognizable format.
//    The String value 1.6e03 is not in a recognizable format.
//    The String value 1.2e-02 is not in a recognizable format.
//    -1326 is outside the range of the UInt16 type.
//    1074122 is outside the range of the UInt16 type.
```

Remarks

Using the ToUInt16(String) method is equivalent to passing `value` to the UInt16.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the UInt16.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(UInt16)

> Important
>
> This API is not CLS-compliant.

Returns the specified 16-bit unsigned integer; no actual conversion is performed.

```
C#

[System.CLSCompliant(false)]
public static ushort ToUInt16 (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to return.

Returns

UInt16

`value` is returned unchanged.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than UInt16.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned integers to an unsigned 16-bit integer.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
ushort result;

foreach (uint number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
```

```
    catch (OverflowException) {
       Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                         number.GetType().Name, number);
    }
}
// The example displays the following output:
//    Converted the UInt32 value '0' to the UInt16 value 0.
//    Converted the UInt32 value '121' to the UInt16 value 121.
//    Converted the UInt32 value '340' to the UInt16 value 340.
//    The UInt32 value 4294967295 is outside the range of the UInt16
type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(String, Int32)

> Important
>
> This API is not CLS-compliant.

Converts the string representation of a number in a specified base to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

ArgumentOutOfRangeException

`value` is Empty.

FormatException

`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example attempts to interpret each element in an array of numeric strings as a hexadecimal value and to convert it to an unsigned 16-bit integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] hexStrings = { "8000", "0FFF", "f000", "00A30", "D",
"-13",
                              "9AC61", "GAD" };
      foreach (string hexString in hexStrings)
      {
         try {
            ushort number = Convert.ToUInt16(hexString, 16);
            Console.WriteLine("Converted '{0}' to {1:N0}.", hexString,
number);
         }
         catch (FormatException) {
            Console.WriteLine("'{0}' is not in the correct format for a
hexadecimal number.",
                              hexString);
         }
         catch (OverflowException) {
            Console.WriteLine("'{0}' is outside the range of an Int16.",
hexString);
         }
         catch (ArgumentException) {
            Console.WriteLine("'{0}' is invalid in base 16.", hexString);
         }
      }
   }
}
// The example displays the following output:
//    Converted '8000' to 32,768.
//    Converted '0FFF' to 4,095.
//    Converted 'f000' to 61,440.
//    Converted '00A30' to 2,608.
//    Converted 'D' to 13.
//    '-13' is invalid in base 16.
//    '9AC61' is outside the range of an Int16.
//    'GAD' is not in the correct format for a hexadecimal number.
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the UInt16 data type supports unsigned values only, the ToUInt16(String, Int32) method assumes that `value` is expressed using unsigned binary representation. In other words, all 16 bits are used to represent the numeric value, and a sign bit is absent. As a result, it is possible to write code in which a signed integer value that is out of the range of the UInt16 data type is converted to a UInt16 value without the method throwing an exception. The following example converts Int16.MinValue to its hexadecimal string representation, and then calls the ToUInt16(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x8000 converts to 32768."

C#

```
// Create a hexadecimal value out of range of the UInt16 type.
string value = Convert.ToString(Int16.MinValue, 16);
// Convert it back to a number.
try
{
   UInt16 number = Convert.ToUInt16(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned short
integer.",
                     value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method or operator is using the appropriate numeric representation to interpret a particular value. The following example illustrates one technique for ensuring that the method does not inappropriately use binary representation to interpret a value that uses two's complement representation when converting a hexadecimal string to a UInt16 value. The example determines whether a value represents a signed or an unsigned integer while it is converting that value to its string representation. When the example converts the value to a UInt16 value, it checks whether the original value was a signed integer. If so, and if its high-order bit is set (which indicates that the original value was negative), the method throws an exception.

C#

```
// Create a negative hexadecimal value out of range of the UInt16 type.
short sourceNumber = Int16.MinValue;
bool isSigned =
Math.Sign((short)sourceNumber.GetType().GetField("MinValue").GetValue(nul
l)) == -1;
string value = Convert.ToString(sourceNumber, 16);
UInt16 targetNumber;
try
{
   targetNumber = Convert.ToUInt16(value, 16);
   if (isSigned && ((targetNumber & 0x8000) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned short
integer.", value);
}
// Displays the following to the console:
//    Unable to convert '0x8000' to an unsigned short integer.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Object, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 16-bit unsigned integer, using the specified culture-specific formatting information.

```
C#

[System.CLSCompliant(false)]
public static ushort ToUInt16 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example defines a `HexString` class that implements the IConvertible interface and that is designed to hold the string representation of both 16-bit signed and 16-bit unsigned values. The class includes a `Sign` property that indicates the sign of its hexadecimal value.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;

public enum SignBit { Negative=-1, Zero=0, Positive=1 };

public struct HexString : IConvertible
{
   private SignBit signBit;
   private string hexString;

   public SignBit Sign
   {
      set { signBit = value; }
      get { return signBit; }
   }

   public string Value
   {
      set {
         if (value.Trim().Length > 4)
            throw new ArgumentException("The string representation of a
160bit integer cannot have more than four characters.");
         else if (! Regex.IsMatch(value, "([0-9,A-F]){1,4}",
RegexOptions.IgnoreCase))
            throw new ArgumentException("The hexadecimal representation
of a 16-bit integer contains invalid characters.");
         else
            hexString = value;
      }
      get { return hexString; }
   }

   // IConvertible implementations.
   public TypeCode GetTypeCode() {
      return TypeCode.Object;
   }

   public bool ToBoolean(IFormatProvider provider)
   {
      return signBit != SignBit.Zero;
   }

   public byte ToByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
```

```
            throw new OverflowException(String.Format("{0} is out of range
of the Byte type.", Convert.ToInt16(hexString, 16)));
         else
            try {
               return Convert.ToByte(UInt16.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
               throw new OverflowException(String.Format("{0} is out of
range of the UInt16 type.", Convert.ToUInt16(hexString, 16)), e);
            }
      }

      public char ToChar(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative) {
            throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToInt16(hexString, 16)));
         }

         UInt16 codePoint = UInt16.Parse(this.hexString,
NumberStyles.HexNumber);
         return Convert.ToChar(codePoint);
      }

      public DateTime ToDateTime(IFormatProvider provider)
      {
         throw new InvalidCastException("Hexadecimal to DateTime conversion
is not supported.");
      }

      public decimal ToDecimal(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
         {
            short hexValue = Int16.Parse(hexString, NumberStyles.HexNumber);
            return Convert.ToDecimal(hexValue);
         }
         else
         {
            ushort hexValue = UInt16.Parse(hexString,
NumberStyles.HexNumber);
            return Convert.ToDecimal(hexValue);
         }
      }

      public double ToDouble(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            return Convert.ToDouble(Int16.Parse(hexString,
NumberStyles.HexNumber));
         else
            return Convert.ToDouble(UInt16.Parse(hexString,
NumberStyles.HexNumber));
      }
```

```
   public short ToInt16(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Int16.Parse(hexString, NumberStyles.HexNumber);
      else
         try {
            return Convert.ToInt16(UInt16.Parse(hexString,
NumberStyles.HexNumber));
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is out of
range of the Int16 type.",

Convert.ToUInt16(hexString, 16)), e);
         }
   }

   public int ToInt32(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToInt32(Int16.Parse(hexString,
NumberStyles.HexNumber));
      else
         return Convert.ToInt32(UInt16.Parse(hexString,
NumberStyles.HexNumber));
   }

   public long ToInt64(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToInt64(Int16.Parse(hexString,
NumberStyles.HexNumber));
      else
         return Int64.Parse(hexString, NumberStyles.HexNumber);
   }

   public sbyte ToSByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         try {
            return Convert.ToSByte(Int16.Parse(hexString,
NumberStyles.HexNumber));
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

Int16.Parse(hexString, NumberStyles.HexNumber), e));
         }
      else
         try {
            return Convert.ToSByte(UInt16.Parse(hexString,
NumberStyles.HexNumber));
         }
```

```
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

UInt16.Parse(hexString, NumberStyles.HexNumber)), e);
         }
   }

   public float ToSingle(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         return Convert.ToSingle(Int16.Parse(hexString,
NumberStyles.HexNumber));
      else
         return Convert.ToSingle(UInt16.Parse(hexString,
NumberStyles.HexNumber));
   }

   public string ToString(IFormatProvider provider)
   {
      return "0x" + this.hexString;
   }

   public object ToType(Type conversionType, IFormatProvider provider)
   {
      switch (Type.GetTypeCode(conversionType))
      {
         case TypeCode.Boolean:
            return this.ToBoolean(null);
         case TypeCode.Byte:
            return this.ToByte(null);
         case TypeCode.Char:
            return this.ToChar(null);
         case TypeCode.DateTime:
            return this.ToDateTime(null);
         case TypeCode.Decimal:
            return this.ToDecimal(null);
         case TypeCode.Double:
            return this.ToDouble(null);
         case TypeCode.Int16:
            return this.ToInt16(null);
         case TypeCode.Int32:
            return this.ToInt32(null);
         case TypeCode.Int64:
            return this.ToInt64(null);
         case TypeCode.Object:
            if (typeof(HexString).Equals(conversionType))
               return this;
            else
               throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.", conversionType.Name));
         case TypeCode.SByte:
            return this.ToSByte(null);
         case TypeCode.Single:
            return this.ToSingle(null);
```

```
            case TypeCode.String:
               return this.ToString(null);
            case TypeCode.UInt16:
               return this.ToUInt16(null);
            case TypeCode.UInt32:
               return this.ToUInt32(null);
            case TypeCode.UInt64:
               return this.ToUInt64(null);
            default:
               throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
         }
      }

      public UInt16 ToUInt16(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt16 type.",
                                                        Int16.Parse(hexString,
NumberStyles.HexNumber)));
         else
            return UInt16.Parse(hexString, NumberStyles.HexNumber);
      }

      public UInt32 ToUInt32(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",
                                                        Int16.Parse(hexString,
NumberStyles.HexNumber)));
         else
            return Convert.ToUInt32(hexString, 16);
      }

      public UInt64 ToUInt64(IFormatProvider provider)
      {
         if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt64 type.",
                                                        Int64.Parse(hexString,
NumberStyles.HexNumber)));
         else
            return Convert.ToUInt64(hexString, 16);
      }
   }
```

The following example shows that a call to the ToUInt16(Object, IFormatProvider) method that passes a `HexString` object as a parameter, in turn, calls the IConvertible.ToUInt16 implementation of the `HexString` class.

C#

```
public class Example
{
   public static void Main()
   {
      ushort positiveValue = 32000;
      short negativeValue = -1;

      HexString positiveString = new HexString();
      positiveString.Sign = (SignBit) Math.Sign(positiveValue);
      positiveString.Value = positiveValue.ToString("X2");

      HexString negativeString = new HexString();
      negativeString.Sign = (SignBit) Math.Sign(negativeValue);
      negativeString.Value = negativeValue.ToString("X2");

      try {
         Console.WriteLine("0x{0} converts to {1}.",
positiveString.Value, Convert.ToUInt16(positiveString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt16
type.",
                           Int16.Parse(negativeString.Value,
NumberStyles.HexNumber));
      }

      try {
         Console.WriteLine("0x{0} converts to {1}.",
negativeString.Value, Convert.ToUInt16(negativeString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt16
type.",
                           Int16.Parse(negativeString.Value,
NumberStyles.HexNumber));
      }
   }
}
// The example displays the following output:
//       0x7D00 converts to 32000.
//       -1 is outside the range of the UInt16 type.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt16 method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number,

`provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 16-bit unsigned integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that recognizes the string "pos" as the positive sign and the string "neg" as the negative sign. It then attempts to convert each element of a numeric string array to an integer using both this provider and the NumberFormatInfo provider for the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set several of its
      // properties that apply to numbers.
      NumberFormatInfo provider = new NumberFormatInfo();
      provider.PositiveSign = "pos ";
      provider.NegativeSign = "neg ";

      // Define an array of strings to convert to UInt16 values.
      string[] values= { "34567", "+34567", "pos 34567", "34567.",
```

```
                          "34567.", "65535", "65535", "65535" };

      foreach (string value in values)
      {
         Console.Write("{0,-12:}  -->  ", value);
         try {
            Console.WriteLine("{0,17}", Convert.ToUInt16(value,
provider));
         }
         catch (FormatException e) {
            Console.WriteLine("{0,17}", e.GetType().Name);
         }
      }
   }
}
// The example displays the following output:
//       34567         -->              34567
//       +34567        -->    FormatException
//       pos 34567     -->              34567
//       34567.        -->    FormatException
//       34567.        -->    FormatException
//       65535         -->              65535
//       65535         -->              65535
//       65535         -->              65535
```

Remarks

The return value is the result of invoking UInt16.Parse on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the UInt16.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Single)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified single-precision floating-point number to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

UInt16

`value`, rounded to the nearest 16-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt16.MaxValue.

Examples

The following example converts each element in an array of String values to an unsigned 16-bit integer.

C#

```
float[] numbers = { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                    0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
ushort result;

foreach (float number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt16 type.",
number);
   }
}
// The example displays the following output:
//    -3.402823E+38 is outside the range of the UInt16 type.
//    -1.38E+10 is outside the range of the UInt16 type.
//    -1023.299 is outside the range of the UInt16 type.
//    -12.98 is outside the range of the UInt16 type.
//    Converted the Single value '0' to the UInt16 value 0.
//    Converted the Single value '9.113E-16' to the UInt16 value 0.
//    Converted the Single value '103.919' to the UInt16 value 104.
//    Converted the Single value '17834.19' to the UInt16 value 17834.
//    3.402823E+38 is outside the range of the UInt16 type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than UInt16.MaxValue.

Examples

The following example attempts to convert each element in an array of unsigned long integers to an unsigned 16-bit integer.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
ushort result;

foreach (ulong number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt64 value '0' to the UInt16 value 0.
//    Converted the UInt64 value '121' to the UInt16 value 121.
//    Converted the UInt64 value '340' to the UInt16 value 340.
//    The UInt64 value 18446744073709551615 is outside the range of the
UInt16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToUInt16(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example converts each element in a signed byte array to an unsigned 16-bit integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
ushort result;

foreach (sbyte number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt16 type.",
number);
   }
}
// The example displays the following output:
//    -128 is outside the range of the UInt16 type.
//    -1 is outside the range of the UInt16 type.
//    Converted the SByte value '0' to the UInt16 value 0.
//    Converted the SByte value '10' to the UInt16 value 10.
//    Converted the SByte value '127' to the UInt16 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Char)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified Unicode character to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

UInt16

The 16-bit unsigned integer equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a character array to an unsigned 16-bit integer.

C#

```
char[] chars = { 'a', 'z', '\x0007', '\x03FF',
                 '\x7FFF', '\xFFFE' };
ushort result;

foreach (char ch in chars)
{
   try {
      result = Convert.ToUInt16(ch);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        ch.GetType().Name, ch,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to convert u+{0} to a UInt16.",
                        ((int)ch).ToString("X4"));
   }
}
// The example displays the following output:
```

```
//    Converted the Char value 'a' to the UInt16 value 97.
//    Converted the Char value 'z' to the UInt16 value 122.
//    Converted the Char value '' to the UInt16 value 7.
//    Converted the Char value '?' to the UInt16 value 1023.
//    Converted the Char value '?' to the UInt16 value 32767.
//    Converted the Char value '?' to the UInt16 value 65534.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Int64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit signed integer to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt16.MaxValue.

Examples

The following example converts each element in a long integer array to an unsigned 16-bit integer.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
ushort result;

foreach (long number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
UInt16 type.
//    The Int64 value -1 is outside the range of the UInt16 type.
//    Converted the Int64 value '0' to the UInt16 value 0.
//    Converted the Int64 value '121' to the UInt16 value 121.
//    Converted the Int64 value '340' to the UInt16 value 340.
//    The Int64 value 9223372036854775807 is outside the range of the
UInt16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Int32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit signed integer to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt16.MaxValue.

Examples

The following example converts each element in an integer array to an unsigned 16-bit integer.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
ushort result;

foreach (int number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int32 value -2147483648 is outside the range of the UInt16
type.
//    The Int32 value -1 is outside the range of the UInt16 type.
//    Converted the Int32 value '0' to the UInt16 value 0.
//    Converted the Int32 value '121' to the UInt16 value 121.
//    Converted the Int32 value '340' to the UInt16 value 340.
//    The Int32 value 2147483647 is outside the range of the UInt16 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Int16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit signed integer to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in a 16-bit integer array to an unsigned 16-bit integer.

C#

```
short[] numbers = { Int16.MinValue, -132, 0, 121, 16103, Int16.MaxValue
};
ushort result;

foreach (short number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int16 value -32768 is outside the range of the UInt16 type.
//    The Int16 value -132 is outside the range of the UInt16 type.
//    Converted the Int16 value '0' to the UInt16 value 0.
//    Converted the Int16 value '121' to the UInt16 value 121.
//    Converted the Int16 value '16103' to the UInt16 value 16103.
//    Converted the Int16 value '32767' to the UInt16 value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Decimal)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified decimal number to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

UInt16

`value`, rounded to the nearest 16-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt16.MaxValue.

Examples

The following example converts each element in an array of Decimal values to an unsigned 16-bit integer.

C#

```
decimal[] numbers = { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                            9214.16m, Decimal.MaxValue };
ushort result;

foreach (decimal number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("{0} is outside the range of the UInt16 type.",
                        number);
   }
}
// The example displays the following output:
//    -79228162514264337593543950335 is outside the range of the UInt16
type.
//    -1034.23 is outside the range of the UInt16 type.
//    -12 is outside the range of the UInt16 type.
//    Converted the Decimal value '0' to the UInt16 value 0.
//    Converted the Decimal value '147' to the UInt16 value 147.
//    Converted the Decimal value '9214.16' to the UInt16 value 9214.
//    79228162514264337593543950335 is outside the range of the UInt16
type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(DateTime)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

```
C#

[System.CLSCompliant(false)]
public static ushort ToUInt16 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

UInt16

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

ToUInt16(Byte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit unsigned integer to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a byte array to an unsigned 16-bit integer.

C#

```
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
ushort result;

foreach (byte byteValue in bytes)
```

```
{
   result = Convert.ToUInt16(byteValue);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     byteValue.GetType().Name, byteValue,
                     result.GetType().Name, result);
}
// The example displays the following output:
//       Converted the Byte value '0' to the UInt16 value 0.
//       Converted the Byte value '14' to the UInt16 value 14.
//       Converted the Byte value '122' to the UInt16 value 122.
//       Converted the Byte value '255' to the UInt16 value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Boolean)

> Important
>
> This API is not CLS-compliant.

Converts the specified Boolean value to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

UInt16

The number 1 if `value` is `true`; otherwise, 0.

Attributes CLSCompliantAttribute

Examples

The following example converts the Boolean values `true` and `false` to unsigned 16-bit integers.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToInt16(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToUInt16(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Object)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

UInt16

A 16-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example attempts to convert each element in an object array to an unsigned 16-bit integer.

C#

```
object[] values= { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                   "104", "103.0", "-1", "1.00e2", "One", 1.00e2};
ushort result;

foreach (object value in values)
{
   try {
      result = Convert.ToUInt16(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt16 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to a UInt16 exists for the {0}
value {1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value 'True' to the UInt16 value 1.
//    The Int32 value -12 is outside the range of the UInt16 type.
//    Converted the Int32 value '163' to the UInt16 value 163.
//    Converted the Int32 value '935' to the UInt16 value 935.
//    Converted the Char value 'x' to the UInt16 value 120.
//    No conversion to a UInt16 exists for the DateTime value 5/12/2009
12:00:00 AM.
//    Converted the String value '104' to the UInt16 value 104.
//    The String value 103.0 is not in a recognizable format.
//    The String value -1 is outside the range of the UInt16 type.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value '100' to the UInt16 value 100.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt16 method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt16(Double)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified double-precision floating-point number to an equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

UInt16

`value`, rounded to the nearest 16-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt16.MaxValue.

Examples

The following example converts each element in an array of Double values to an unsigned 16-bit integer.

C#

```
double[] numbers = { Double.MinValue, -1.38e10, -1023.299, -12.98,
                     0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
ushort result;

foreach (double number in numbers)
{
   try {
      result = Convert.ToUInt16(number);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException)
   {
      Console.WriteLine("{0} is outside the range of the UInt16 type.",
number);
   }
}
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the UInt16 type.
//    -13800000000 is outside the range of the UInt16 type.
//    -1023.299 is outside the range of the UInt16 type.
//    -12.98 is outside the range of the UInt16 type.
//    Converted the Double value '0' to the UInt16 value 0.
//    Converted the Double value '9.113E-16' to the UInt16 value 0.
//    Converted the Double value '103.919' to the UInt16 value 104.
```

```
//    Converted the Double value '17834.191' to the UInt16 value 17834.
//    1.79769313486232E+308 is outside the range of the UInt16 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToUInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 32-bit unsigned integer.

Overloads

Expand table

ToUInt32(String)	Converts the specified string representation of a number to an equivalent 32-bit unsigned integer.
ToUInt32(UInt64)	Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit unsigned integer.
ToUInt32(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit unsigned integer.
ToUInt32(UInt32)	Returns the specified 32-bit unsigned integer; no actual conversion is performed.
ToUInt32(DateTime)	Calling this method always throws InvalidCastException.
ToUInt32(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 32-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt32(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 32-bit unsigned integer.
ToUInt32(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 32-bit unsigned integer.
ToUInt32(Object, IFormatProvider)	Converts the value of the specified object to a 32-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt32(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 32-bit unsigned integer.
ToUInt32(Char)	Converts the value of the specified Unicode character to the

	equivalent 32-bit unsigned integer.
ToUInt32(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 32-bit unsigned integer.
ToUInt32(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 32-bit unsigned integer.
ToUInt32(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 32-bit unsigned integer.
ToUInt32(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 32-bit unsigned integer.
ToUInt32(Decimal)	Converts the value of the specified decimal number to an equivalent 32-bit unsigned integer.
ToUInt32(Object)	Converts the value of the specified object to a 32-bit unsigned integer.
ToUInt32(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit unsigned integer.
ToUInt32(Boolean)	Converts the specified Boolean value to the equivalent 32-bit unsigned integer.

ToUInt32(String)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example interprets the elements in a string array as numeric strings and attempts to convert them to unsigned integers.

C#

```
string[] values = { "One", "1.34e28", "-26.87", "-18", "-6.00",
                    " 0", "137", "1601.9", Int32.MaxValue.ToString() };
uint result;

foreach (string value in values)
{
   try {
      result = Convert.ToUInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value '{1}' is outside the range of the
UInt32 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value '{1}' is not in a recognizable
format.",
```

```
                                value.GetType().Name, value);
    }
}
// The example displays the following output:
//    The String value 'One' is not in a recognizable format.
//    The String value '1.34e28' is not in a recognizable format.
//    The String value '-26.87' is not in a recognizable format.
//    The String value '-18' is outside the range of the UInt32 type.
//    The String value '-6.00' is not in a recognizable format.
//    Converted the String value ' 0' to the UInt32 value 0.
//    Converted the String value '137' to the UInt32 value 137.
//    The String value '1601.9' is not in a recognizable format.
//    Converted the String value '2147483647' to the UInt32 value
2147483647.
```

Remarks

Using the ToUInt32(String) method is equivalent to passing `value` to the UInt32.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the UInt32.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(UInt64)

Important

This API is not CLS-compliant.

Converts the value of the specified 64-bit unsigned integer to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (ulong value);
```

Parameters

value UInt64

The 64-bit unsigned integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in an unsigned long integer array to an unsigned integer value.

C#

```
ulong[] numbers = { UInt64.MinValue, 121, 340, UInt64.MaxValue };
uint result;
foreach (ulong number in numbers)
{
   try {
      result = Convert.ToUInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
```

```
                        result.GetType().Name, result);
    }
    catch (OverflowException) {
       Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        number.GetType().Name, number);
    }
}
// The example displays the following output:
//    Converted the UInt64 value 0 to the UInt32 value 0.
//    Converted the UInt64 value 121 to the UInt32 value 121.
//    Converted the UInt64 value 340 to the UInt32 value 340.
//    The UInt64 value 18446744073709551615 is outside the range of the
UInt32 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an unsigned 16-bit integer array to an unsigned integer value.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
uint result;
foreach (ushort number in numbers)
{
   result = Convert.ToUInt32(number);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     number.GetType().Name, number,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the UInt32 value 0.
//    Converted the UInt16 value 121 to the UInt32 value 121.
//    Converted the UInt16 value 340 to the UInt32 value 340.
//    Converted the UInt16 value 65535 to the UInt32 value 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(UInt32)

> Important
>
> This API is not CLS-compliant.

Returns the specified 32-bit unsigned integer; no actual conversion is performed.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to return.

Returns

UInt32

`value` is returned unchanged.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(DateTime)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

UInt32

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToUInt32(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 32-bit unsigned integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that recognizes the string "pos" as the positive sign and the string "neg" as the negative sign. It then attempts to convert each element of a numeric string array to an integer using both this provider and the NumberFormatInfo provider for the invariant culture.

C#

```
using System;
using System.Globalization;

public class Class1
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set several of its
      // properties that apply to numbers.
      NumberFormatInfo provider = new NumberFormatInfo();
      provider.PositiveSign = "pos ";
      provider.NegativeSign = "neg ";

      // Define an array of numeric strings.
      string[] values = { "123456789", "+123456789", "pos 123456789",
                          "123456789.", "123,456,789",  "4294967295",
                          "4294967296", "-1", "neg 1" };

      foreach (string value in values)
      {
         Console.Write("{0,-20} -->", value);
         try {
            Console.WriteLine("{0,20}", Convert.ToUInt32(value,
provider));
         }
         catch (FormatException) {
            Console.WriteLine("{0,20}", "Bad Format");
         }
         catch (OverflowException) {
            Console.WriteLine("{0,20}", "Numeric Overflow");
```

```
            }
        }
    }
}
// The example displays the following output:
//       123456789            -->           123456789
//       +123456789           -->          Bad Format
//       pos 123456789        -->           123456789
//       123456789.           -->          Bad Format
//       123,456,789          -->          Bad Format
//       4294967295           -->          4294967295
//       4294967296           -->    Numeric Overflow
//       -1                   -->          Bad Format
//       neg 1                -->    Numeric Overflow
```

Remarks

The return value is the result of invoking UInt32.Parse on `value`.

`provider` is an IFormatProvider instance that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the UInt32.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(String, Int32)

> Important
>
> This API is not CLS-compliant.

Converts the string representation of a number in a specified base to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

ArgumentOutOfRangeException

`value` is Empty.

FormatException

`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example attempts to interpret each element in an array of numeric strings as a hexadecimal value and to convert it to an unsigned integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] hexStrings = { "80000000", "0FFFFFFF", "F0000000",
"00A3000", "D",
                              "-13", "9AC61", "GAD", "FFFFFFFFFF" };

      foreach (string hexString in hexStrings)
      {
         Console.Write("{0,-12}  -->  ", hexString);
         try {
            uint number = Convert.ToUInt32(hexString, 16);
            Console.WriteLine("{0,18:N0}", number);
         }
         catch (FormatException) {
            Console.WriteLine("{0,18}", "Bad Format");
         }
         catch (OverflowException)
         {
            Console.WriteLine("{0,18}", "Numeric Overflow");
         }
         catch (ArgumentException) {
            Console.WriteLine("{0,18}", "Invalid in Base 16");
```

```
            }
        }
    }
}
// The example displays the following output:
//       80000000      -->       2,147,483,648
//       0FFFFFFF      -->         268,435,455
//       F0000000      -->       4,026,531,840
//       00A3000       -->             667,648
//       D             -->                  13
//       -13           -->  Invalid in Base 16
//       9AC61         -->             633,953
//       GAD           -->          Bad Format
//       FFFFFFFFFF    -->    Numeric Overflow
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the UInt32 data type supports unsigned values only, the ToUInt32(String, Int32) method assumes that `value` is expressed using unsigned binary representation. In other words, all 32 bits are used to represent the numeric value, and a sign bit is absent. As a result, it is possible to write code in which a signed integer value that is out of the range of the UInt32 data type is converted to a UInt32 value without the method throwing an exception. The following example converts MinValue to its hexadecimal string representation, and then calls the ToUInt32(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x80000000 converts to 2147483648."

C#

```
// Create a hexadecimal value out of range of the UInt32 type.
string value = Convert.ToString(Int32.MinValue, 16);
// Convert it back to a number.
try
{
   UInt32 number = Convert.ToUInt32(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned integer.",
                     value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method or operator is using the appropriate numeric representation to interpret a particular value. The following example illustrates one technique for ensuring that the method does not inappropriately use binary representation to interpret a value that uses two's complement representation when converting a hexadecimal string to a UInt32 value. The example determines whether a value represents a signed or an unsigned integer while it is converting that value to its string representation. When the example converts the value to a UInt32 value, it checks whether the original value was a signed integer. If so, and if its high-order bit is set (which indicates that the original value was negative), the method throws an exception.

C#

```
// Create a negative hexadecimal value out of range of the UInt32 type.
int sourceNumber = Int32.MinValue;
bool isSigned =
Math.Sign((int)sourceNumber.GetType().GetField("MinValue").GetValue(null)
) == -1;
string value = Convert.ToString(sourceNumber, 16);
UInt32 targetNumber;
try
{
   targetNumber = Convert.ToUInt32(value, 16);
   if (isSigned && ((targetNumber & 0x80000000) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned integer.",
                     value);
}
// Displays the following to the console:
//    Unable to convert '0x80000000' to an unsigned integer.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Single)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified single-precision floating-point number to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

UInt32

`value`, rounded to the nearest 32-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in an array of Single values to an unsigned integer.

C#

```
float[] values= { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                  0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
uint result;

foreach (float value in values)
{
   try {
      result = Convert.ToUInt32(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    The Single value -3.402823E+38 is outside the range of the UInt32
type.
//    The Single value -1.38E+10 is outside the range of the UInt32 type.
//    The Single value -1023.299 is outside the range of the UInt32 type.
//    The Single value -12.98 is outside the range of the UInt32 type.
//    Converted the Single value 0 to the UInt32 value 0.
//    Converted the Single value 9.113E-16 to the UInt32 value 0.
//    Converted the Single value 103.919 to the UInt32 value 104.
//    Converted the Single value 17834.19 to the UInt32 value 17834.
//    The Single value 3.402823E+38 is outside the range of the UInt32
type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Object, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 32-bit unsigned integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example defines a `HexString` class that implements the IConvertible interface and that is designed to hold the string representation of both 32-bit signed and 32-bit unsigned values. The class includes a `Sign` property that indicates the sign of its hexadecimal value.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;

public enum SignBit { Negative=-1, Zero=0, Positive=1 };

public struct HexString : IConvertible
{
   private SignBit signBit;
   private string hexString;

   public SignBit Sign
   {
      set { signBit = value; }
      get { return signBit; }
   }

   public string Value
   {
```

```
      set {
         if (value.Trim().Length > 8)
            throw new ArgumentException("The string representation of a
32-bit integer cannot have more than 8 characters.");
         else if (! Regex.IsMatch(value, "([0-9,A-F]){1,8}",
RegexOptions.IgnoreCase))
            throw new ArgumentException("The hexadecimal representation
of a 32-bit integer contains invalid characters.");
         else
            hexString = value;
      }
      get { return hexString; }
   }

   // IConvertible implementations.
   public TypeCode GetTypeCode()
   {
      return TypeCode.Object;
   }

   public bool ToBoolean(IFormatProvider provider)
   {
      return signBit != SignBit.Zero;
   }

   public byte ToByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is out of range
of the Byte type.", Convert.ToInt32(hexString, 16)));
      else
         try {
            return Byte.Parse(hexString, NumberStyles.HexNumber);
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is out of
range of the Byte type.", Convert.ToUInt32(hexString, 16)), e);
         }
   }

   public char ToChar(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToInt32(hexString, 16)));

      try {
         ushort codePoint = UInt16.Parse(this.hexString,
NumberStyles.HexNumber);
         return Convert.ToChar(codePoint);
      }
      catch (OverflowException e) {
         throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToUInt32(hexString, 16)), e);
      }
```

```
    }

    public DateTime ToDateTime(IFormatProvider provider)
    {
        throw new InvalidCastException("Hexadecimal to DateTime conversion
is not supported.");
    }

    public decimal ToDecimal(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
        {
            int hexValue = Int32.Parse(hexString, NumberStyles.HexNumber);
            return Convert.ToDecimal(hexValue);
        }
        else
        {
            uint hexValue = UInt32.Parse(hexString, NumberStyles.HexNumber);
            return Convert.ToDecimal(hexValue);
        }
    }

    public double ToDouble(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToDouble(Int32.Parse(hexString,
NumberStyles.HexNumber));
        else
            return Convert.ToDouble(UInt32.Parse(hexString,
NumberStyles.HexNumber));
    }

    public short ToInt16(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            try {
                return Convert.ToInt16(Int32.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int16 type.", Convert.ToInt32(hexString, 16)), e);
            }
        else
            try {
                return Convert.ToInt16(UInt32.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int16 type.", Convert.ToUInt32(hexString, 16)), e);
            }
    }

    public int ToInt32(IFormatProvider provider)
```

```
    {
        if (signBit == SignBit.Negative)
            return Int32.Parse(hexString, NumberStyles.HexNumber);
        else
            try {
                return Convert.ToInt32(UInt32.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int32 type.", Convert.ToUInt32(hexString, 16)), e);
            }
    }

    public long ToInt64(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToInt64(Int32.Parse(hexString,
NumberStyles.HexNumber));
        else
            return Int64.Parse(hexString, NumberStyles.HexNumber);
    }

    public sbyte ToSByte(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            try {
                return Convert.ToSByte(Int32.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

Int32.Parse(hexString, NumberStyles.HexNumber), e));
            }
        else
            try {
                return Convert.ToSByte(UInt32.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

UInt32.Parse(hexString, NumberStyles.HexNumber)), e);
            }
    }

    public float ToSingle(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToSingle(Int32.Parse(hexString,
NumberStyles.HexNumber));
        else
```

```
            return Convert.ToSingle(UInt32.Parse(hexString,
NumberStyles.HexNumber));
        }

        public string ToString(IFormatProvider provider)
        {
            return "0x" + this.hexString;
        }

        public object ToType(Type conversionType, IFormatProvider provider)
        {
            switch (Type.GetTypeCode(conversionType))
            {
                case TypeCode.Boolean:
                    return this.ToBoolean(null);
                case TypeCode.Byte:
                    return this.ToByte(null);
                case TypeCode.Char:
                    return this.ToChar(null);
                case TypeCode.DateTime:
                    return this.ToDateTime(null);
                case TypeCode.Decimal:
                    return this.ToDecimal(null);
                case TypeCode.Double:
                    return this.ToDouble(null);
                case TypeCode.Int16:
                    return this.ToInt16(null);
                case TypeCode.Int32:
                    return this.ToInt32(null);
                case TypeCode.Int64:
                    return this.ToInt64(null);
                case TypeCode.Object:
                    if (typeof(HexString).Equals(conversionType))
                        return this;
                    else
                        throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.", conversionType.Name));
                case TypeCode.SByte:
                    return this.ToSByte(null);
                case TypeCode.Single:
                    return this.ToSingle(null);
                case TypeCode.String:
                    return this.ToString(null);
                case TypeCode.UInt16:
                    return this.ToUInt16(null);
                case TypeCode.UInt32:
                    return this.ToUInt32(null);
                case TypeCode.UInt64:
                    return this.ToUInt64(null);
                default:
                    throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
            }
        }
```

```
   public ushort ToUInt16(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt16 type.",
                                                     Int32.Parse(hexString,
NumberStyles.HexNumber)));
      else
         try {
            return Convert.ToUInt16(UInt32.Parse(hexString,
NumberStyles.HexNumber));
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is out of
range of the UInt16 type.", Convert.ToUInt32(hexString, 16)), e);
         }
   }

   public uint ToUInt32(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",
                                                     Int32.Parse(hexString,
NumberStyles.HexNumber)));
      else
         return Convert.ToUInt32(hexString, 16);
   }

   public ulong ToUInt64(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt64 type.",
                                                     Int32.Parse(hexString,
NumberStyles.HexNumber)));
      else
         return Convert.ToUInt64(hexString, 16);
   }
}
```

The following example shows that a call to the ToUInt32(Object, IFormatProvider) method that passes a `HexString` object as a parameter, in turn, calls the IConvertible.ToUInt32 implementation of the `HexString` class.

C#

```
public class Example
{
   public static void Main()
   {
      uint positiveValue = 320000000;
```

```
      int negativeValue = -1;

      HexString positiveString = new HexString();
      positiveString.Sign = (SignBit) Math.Sign(positiveValue);
      positiveString.Value = positiveValue.ToString("X4");

      HexString negativeString = new HexString();
      negativeString.Sign = (SignBit) Math.Sign(negativeValue);
      negativeString.Value = negativeValue.ToString("X4");

      try {
         Console.WriteLine("0x{0} converts to {1}.",
positiveString.Value, Convert.ToUInt32(positiveString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt32
type.",
                           Int32.Parse(positiveString.Value,
NumberStyles.HexNumber));
      }

      try {
         Console.WriteLine("0x{0} converts to {1}.",
negativeString.Value, Convert.ToUInt32(negativeString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt32
type.",
                           Int32.Parse(negativeString.Value,
NumberStyles.HexNumber));
      }
   }
}
// The example dosplays the following output:
//       0x1312D000 converts to 320000000.
//       -1 is outside the range of the UInt32 type.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt32 method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number, `provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in a signed byte array to an unsigned integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
uint result;

foreach (sbyte number in numbers)
{
   try {
      result = Convert.ToUInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The SByte value -128 is outside the range of the UInt32 type.
//    The SByte value -1 is outside the range of the UInt32 type.
//    Converted the SByte value 0 to the UInt32 value 0.
//    Converted the SByte value 10 to the UInt32 value 10.
//    Converted the SByte value 127 to the UInt32 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Char)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified Unicode character to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a character array to an unsigned integer.

C#

```
char[] chars = { 'a', 'z', '\u0007', '\u03FF',
                 '\u7FFF', '\uFFFE' };
```

```
uint result;

foreach (char ch in chars)
{
   result = Convert.ToUInt32(ch);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     ch.GetType().Name, ch,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Char value 'a' to the UInt32 value 97.
//    Converted the Char value 'z' to the UInt32 value 122.
//    Converted the Char value '•' to the UInt32 value 7.
//    Converted the Char value 'Ɔ' to the UInt32 value 1023.
//    Converted the Char value '翻' to the UInt32 value 32767.
//    Converted the Char value '￾' to the UInt32 value 65534.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Int64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit signed integer to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in a long integer array to an unsigned integer.

C#

```
long[] numbers = { Int64.MinValue, -1, 0, 121, 340, Int64.MaxValue };
uint result;
foreach (long number in numbers)
{
   try {
      result = Convert.ToUInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        number.GetType().Name, number);
   }
}
```

```
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
UInt32 type.
//    The Int64 value -1 is outside the range of the UInt32 type.
//    Converted the Int64 value 0 to the UInt32 value 0.
//    Converted the Int64 value 121 to the UInt32 value 121.
//    Converted the Int64 value 340 to the UInt32 value 340.
//    The Int64 value 9223372036854775807 is outside the range of the
UInt32 type.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Int32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit signed integer to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in an integer array to an unsigned integer.

C#

```
int[] numbers = { Int32.MinValue, -1203, 0, 121, 1340, Int32.MaxValue };
uint result;
foreach (int number in numbers)
{
   try {
      result = Convert.ToUInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int32 value -2147483648 is outside the range of the UInt32
type.
//    The Int32 value -1203 is outside the range of the UInt32 type.
//    Converted the Int32 value 0 to the UInt32 value 0.
//    Converted the Int32 value 121 to the UInt32 value 121.
//    Converted the Int32 value 1340 to the UInt32 value 1340.
//    Converted the Int32 value 2147483647 to the UInt32 value
2147483647.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Int16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit signed integer to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in a 16-bit integer array to an unsigned integer.

C#

```
short[] numbers= { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
uint result;

foreach (short number in numbers)
{
   try {
      result = Convert.ToUInt32(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                           number.GetType().Name, number,
                           result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int16 value -32768 is outside the range of the UInt32 type.
//    The Int16 value -1 is outside the range of the UInt32 type.
//    Converted the Int16 value 0 to the UInt32 value 0.
//    Converted the Int16 value 121 to the UInt32 value 121.
//    Converted the Int16 value 340 to the UInt32 value 340.
//    Converted the Int16 value 32767 to the UInt32 value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Double)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified double-precision floating-point number to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

UInt32

`value`, rounded to the nearest 32-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in an array of Double values to an unsigned integer.

C#

```
double[] values= { Double.MinValue, -1.38e10, -1023.299, -12.98,
                   0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
uint result;

foreach (double value in values)
{
   try {
      result = Convert.ToUInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    The Double value -1.79769313486232E+308 is outside the range of the
UInt32 type.
//    The Double value -13800000000 is outside the range of the UInt32
type.
//    The Double value -1023.299 is outside the range of the UInt32 type.
//    The Double value -12.98 is outside the range of the UInt32 type.
//    Converted the Double value '0' to the UInt32 value 0.
//    Converted the Double value '9.113E-16' to the UInt32 value 0.
//    Converted the Double value '103.919' to the UInt32 value 104.
//    Converted the Double value '17834.191' to the UInt32 value 17834.
//    The Double value 1.79769313486232E+308 is outside the range of the
UInt32 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Decimal)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified decimal number to an equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

UInt32

`value`, rounded to the nearest 32-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in an array of Decimal values to an unsigned integer.

C#

```
decimal[] values= { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                    199.55m, 9214.16m, Decimal.MaxValue };
uint result;

foreach (decimal value in values)
{
   try {
      result = Convert.ToUInt32(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt32 type.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    The Decimal value -79228162514264337593543950335 is outside the
range of the UInt32 type.
//    The Decimal value -1034.23 is outside the range of the UInt32 type.
//    The Decimal value -12 is outside the range of the UInt32 type.
//    Converted the Decimal value '0' to the UInt32 value 0.
//    Converted the Decimal value '147' to the UInt32 value 147.
//    Converted the Decimal value '199.55' to the UInt32 value 200.
//    Converted the Decimal value '9214.16' to the UInt32 value 9214.
//    The Decimal value 79228162514264337593543950335 is outside the
range of the UInt32 type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Object)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example attempts to convert each element in an object array to an unsigned integer.

C#

```
object[] values = { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                    "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2, 16.3e42};
uint result;

foreach (object value in values)
{
   try {
      result = Convert.ToUInt32(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value '{1}' is outside the range of the
UInt32 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to a UInt32 exists for the {0}
value '{1}'.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value True to the UInt32 value 1.
```

```
//    The Int32 value '-12' is outside the range of the UInt32 type.
//    Converted the Int32 value 163 to the UInt32 value 163.
//    Converted the Int32 value 935 to the UInt32 value 935.
//    Converted the Char value x to the UInt32 value 120.
//    No conversion to a UInt32 exists for the DateTime value '5/12/2009
12:00:00 AM'.
//    Converted the String value 104 to the UInt32 value 104.
//    The String value 103.0 is not in a recognizable format.
//    The String value '-1' is outside the range of the UInt32 type.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the UInt32 value 100.
//    The Double value '1.63E+43' is outside the range of the UInt32
type.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt32 method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Byte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit unsigned integer to the equivalent 32-bit unsigned integer.

```C#
[System.CLSCompliant(false)]
public static uint ToUInt32 (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

UInt32

A 32-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a byte array to an unsigned integer.

```C#
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
uint result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToUInt32(byteValue);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     byteValue.GetType().Name, byteValue,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Byte value 0 to the UInt32 value 0.
//    Converted the Byte value 14 to the UInt32 value 14.
//    Converted the Byte value 122 to the UInt32 value 122.
//    Converted the Byte value 255 to the UInt32 value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt32(Boolean)

> Important
>
> This API is not CLS-compliant.

Converts the specified Boolean value to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

UInt32

The number 1 if `value` is `true`; otherwise, 0.

Attributes CLSCompliantAttribute

Examples

The following example converts the Boolean values `true` and `false` to unsigned integers.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToUInt32(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToUInt32(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.ToUInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified value to a 64-bit unsigned integer.

Overloads

Expand table

ToUInt64(String)	Converts the specified string representation of a number to an equivalent 64-bit unsigned integer.
ToUInt64(UInt16)	Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit unsigned integer.
ToUInt64(UInt32)	Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit unsigned integer.
ToUInt64(String, IFormatProvider)	Converts the specified string representation of a number to an equivalent 64-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt64(Object, IFormatProvider)	Converts the value of the specified object to a 64-bit unsigned integer, using the specified culture-specific formatting information.
ToUInt64(Single)	Converts the value of the specified single-precision floating-point number to an equivalent 64-bit unsigned integer.
ToUInt64(String, Int32)	Converts the string representation of a number in a specified base to an equivalent 64-bit unsigned integer.
ToUInt64(UInt64)	Returns the specified 64-bit unsigned integer; no actual conversion is performed.
ToUInt64(SByte)	Converts the value of the specified 8-bit signed integer to the equivalent 64-bit unsigned integer.
ToUInt64(Int16)	Converts the value of the specified 16-bit signed integer to the equivalent 64-bit unsigned integer.

ToUInt64(Int64)	Converts the value of the specified 64-bit signed integer to an equivalent 64-bit unsigned integer.
ToUInt64(Int32)	Converts the value of the specified 32-bit signed integer to an equivalent 64-bit unsigned integer.
ToUInt64(Double)	Converts the value of the specified double-precision floating-point number to an equivalent 64-bit unsigned integer.
ToUInt64(Decimal)	Converts the value of the specified decimal number to an equivalent 64-bit unsigned integer.
ToUInt64(DateTime)	Calling this method always throws InvalidCastException.
ToUInt64(Char)	Converts the value of the specified Unicode character to the equivalent 64-bit unsigned integer.
ToUInt64(Byte)	Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit unsigned integer.
ToUInt64(Boolean)	Converts the specified Boolean value to the equivalent 64-bit unsigned integer.
ToUInt64(Object)	Converts the value of the specified object to a 64-bit unsigned integer.

ToUInt64(String)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (string? value);
```

Parameters

`value` String

A string that contains the number to convert.

Returns

UInt64

A 64-bit signed integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example interprets the elements of a string array as numeric strings and attempts to convert them to unsigned long integers.

C#

```
string[] values = { "One", "1.34e28", "-26.87", "-18", "-6.00",
                    " 0", "137", "1601.9", Int32.MaxValue.ToString() };
ulong result;

foreach (string value in values)
{
   try {
      result = Convert.ToUInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt64 type.",
value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value '{1}' is not in a recognizable
format.",
                        value.GetType().Name, value);
```

```
    }
}
// The example displays the following output:
//    The String value 'One' is not in a recognizable format.
//    The String value '1.34e28' is not in a recognizable format.
//    The String value '-26.87' is not in a recognizable format.
//    -18 is outside the range of the UInt64 type.
//    The String value '-6.00' is not in a recognizable format.
//    Converted the String value ' 0' to the UInt64 value 0.
//    Converted the String value '137' to the UInt64 value 137.
//    The String value '1601.9' is not in a recognizable format.
//    Converted the String value '2147483647' to the UInt64 value
2147483647.
```

Remarks

Using the ToInt64(String) method is equivalent to passing `value` to the Int64.Parse(String) method. `value` is interpreted by using the formatting conventions of the current culture.

If you prefer not to handle an exception if the conversion fails, you can call the UInt64.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(UInt16)

Important

This API is not CLS-compliant.

Converts the value of the specified 16-bit unsigned integer to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (ushort value);
```

Parameters

value UInt16

The 16-bit unsigned integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an array of unsigned 16-bit integers to an unsigned long integer.

C#

```
ushort[] numbers = { UInt16.MinValue, 121, 340, UInt16.MaxValue };
ulong result;

foreach (ushort number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
```

```
                        number.GetType().Name, number);
    }
}
// The example displays the following output:
//    Converted the UInt16 value 0 to the UInt64 value 0.
//    Converted the UInt16 value 121 to the UInt64 value 121.
//    Converted the UInt16 value 340 to the UInt64 value 340.
//    Converted the UInt16 value 65535 to the UInt64 value 65535.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit unsigned integer to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in an unsigned integer array to an unsigned long integer.

C#

```
uint[] numbers = { UInt32.MinValue, 121, 340, UInt32.MaxValue };
ulong result;

foreach (uint number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    Converted the UInt32 value 0 to the UInt64 value 0.
//    Converted the UInt32 value 121 to the UInt64 value 121.
//    Converted the UInt32 value 340 to the UInt64 value 340.
//    Converted the UInt32 value 4294967295 to the UInt64 value
4294967295.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the specified string representation of a number to an equivalent 64-bit unsigned integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (string? value, IFormatProvider? provider);
```

Parameters

`value` String

A string that contains the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` does not consist of an optional sign followed by a sequence of digits (0 through 9).

OverflowException

`value` represents a number that is less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that recognizes the string "pos" as the positive sign and the string "neg" as the negative sign. It then attempts to convert each element of a numeric string array to an unsigned long integer. The conversion uses both the custom provider and the NumberFormatInfo provider for the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Create a NumberFormatInfo object and set several properties.
      NumberFormatInfo provider = new NumberFormatInfo();
      provider.PositiveSign = "pos ";
      provider.NegativeSign = "neg ";

      // Define an array of numeric strings.
      string[] values = { "123456789012", "+123456789012",
                          "pos 123456789012", "123456789012.",
                          "123,456,789,012", "18446744073709551615",
                          "18446744073709551616", "neg 1", "-1" };
      //  Convert the strings using the format provider.
      foreach (string value in values)
      {
         Console.Write("{0,-20}  -->  ", value);
         try {
            Console.WriteLine("{0,20}", Convert.ToUInt64(value,
provider));
         }
         catch (FormatException) {
            Console.WriteLine("{0,20}", "Invalid Format");
         }
         catch (OverflowException) {
            Console.WriteLine("{0,20}", "Numeric Overflow");
         }
```

```
        }
    }
}
// The example displays the following output:
//    123456789012          -->          123456789012
//    +123456789012         -->        Invalid Format
//    pos 123456789012      -->          123456789012
//    123456789012.         -->        Invalid Format
//    123,456,789,012       -->        Invalid Format
//    18446744073709551615  --> 18446744073709551615
//    18446744073709551616  -->      Numeric Overflow
//    neg 1                 -->      Numeric Overflow
//    -1                    -->        Invalid Format
```

Remarks

The return value is the result of invoking UInt64.Parse on `value`.

`provider` is an IFormatProvider implementation that obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `value`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

If you prefer not to handle an exception if the conversion fails, you can call the UInt64.TryParse method instead. It returns a Boolean value that indicates whether the conversion succeeded or failed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Object, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 64-bit unsigned integer, using the specified culture-specific formatting information.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (object? value, IFormatProvider? provider);
```

Parameters

`value` Object

An object that implements the IConvertible interface.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example defines a `HexString` class that implements the IConvertible interface and that is designed to hold the string representation of both 64-bit signed and 64-bit unsigned values. The class includes a `Sign` property that indicates the sign of its hexadecimal value.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;

public enum SignBit { Negative=-1, Zero=0, Positive=1 };

public struct HexString : IConvertible
{
   private SignBit signBit;
   private string hexString;

   public SignBit Sign
   {
      set { signBit = value; }
      get { return signBit; }
   }

   public string Value
   {
      set
      {
         if (value.Trim().Length > 16)
            throw new ArgumentException("The hexadecimal representation
of a 64-bit integer cannot have more than 16 characters.");
         else if (! Regex.IsMatch(value, "([0-9,A-F]){1,8}",
RegexOptions.IgnoreCase))
            throw new ArgumentException("The hexadecimal representation
of a 64-bit integer contains invalid characters.");
         else
            hexString = value;
      }
      get { return hexString; }
   }

   // IConvertible implementations.
   public TypeCode GetTypeCode()
   {
      return TypeCode.Object;
   }
```

```
   public bool ToBoolean(IFormatProvider provider)
   {
      return signBit != SignBit.Zero;
   }

   public byte ToByte(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is out of range
of the Byte type.", Convert.ToInt64(hexString, 16)));
      else
         try {
            return Byte.Parse(hexString, NumberStyles.HexNumber);
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is out of
range of the Byte type.", Convert.ToUInt64(hexString, 16)), e);
         }
   }

   public char ToChar(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToInt64(hexString, 16)));

      try {
         ushort codePoint = UInt16.Parse(this.hexString,
NumberStyles.HexNumber);
         return Convert.ToChar(codePoint);
      }
      catch (OverflowException) {
         throw new OverflowException(String.Format("{0} is out of range
of the Char type.", Convert.ToUInt64(hexString, 16)));
      }
   }

   public DateTime ToDateTime(IFormatProvider provider)
   {
      throw new InvalidCastException("Hexadecimal to DateTime conversion
is not supported.");
   }

   public decimal ToDecimal(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
      {
         long hexValue = Int64.Parse(hexString, NumberStyles.HexNumber);
         return Convert.ToDecimal(hexValue);
      }
      else
      {
         ulong hexValue = UInt64.Parse(hexString,
NumberStyles.HexNumber);
```

```
            return Convert.ToDecimal(hexValue);
        }
    }

    public double ToDouble(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToDouble(Int64.Parse(hexString,
NumberStyles.HexNumber));
        else
            return Convert.ToDouble(UInt64.Parse(hexString,
NumberStyles.HexNumber));
    }

    public short ToInt16(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            try {
                return Convert.ToInt16(Int64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int16 type.", Convert.ToInt64(hexString, 16)), e);
            }
        else
            try {
                return Convert.ToInt16(UInt64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int16 type.", Convert.ToUInt64(hexString, 16)), e);
            }
    }

    public int ToInt32(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            try {
                return Convert.ToInt32(Int64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int32 type.", Convert.ToUInt64(hexString, 16)), e);
            }
        else
            try {
                return Convert.ToInt32(UInt64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int32 type.", Convert.ToUInt64(hexString, 16)), e);
```

```
            }
    }

    public long ToInt64(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Int64.Parse(hexString, NumberStyles.HexNumber);
        else
            try {
                return Convert.ToInt64(UInt64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is out of
range of the Int64 type.", Convert.ToUInt64(hexString, 16)), e);
            }
    }

    public sbyte ToSByte(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            try {
                return Convert.ToSByte(Int64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

Int64.Parse(hexString, NumberStyles.HexNumber), e));
            }
        else
            try {
                return Convert.ToSByte(UInt64.Parse(hexString,
NumberStyles.HexNumber));
            }
            catch (OverflowException e) {
                throw new OverflowException(String.Format("{0} is outside the
range of the SByte type.",

UInt64.Parse(hexString, NumberStyles.HexNumber)), e);
            }
    }

    public float ToSingle(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            return Convert.ToSingle(Int64.Parse(hexString,
NumberStyles.HexNumber));
        else
            return Convert.ToSingle(UInt64.Parse(hexString,
NumberStyles.HexNumber));
    }

    public string ToString(IFormatProvider provider)
```

```
    {
        return "0x" + this.hexString;
    }

    public object ToType(Type conversionType, IFormatProvider provider)
    {
        switch (Type.GetTypeCode(conversionType))
        {
            case TypeCode.Boolean:
                return this.ToBoolean(null);
            case TypeCode.Byte:
                return this.ToByte(null);
            case TypeCode.Char:
                return this.ToChar(null);
            case TypeCode.DateTime:
                return this.ToDateTime(null);
            case TypeCode.Decimal:
                return this.ToDecimal(null);
            case TypeCode.Double:
                return this.ToDouble(null);
            case TypeCode.Int16:
                return this.ToInt16(null);
            case TypeCode.Int32:
                return this.ToInt32(null);
            case TypeCode.Int64:
                return this.ToInt64(null);
            case TypeCode.Object:
                if (typeof(HexString).Equals(conversionType))
                    return this;
                else
                    throw new InvalidCastException(String.Format("Conversion
to a {0} is not supported.", conversionType.Name));
            case TypeCode.SByte:
                return this.ToSByte(null);
            case TypeCode.Single:
                return this.ToSingle(null);
            case TypeCode.String:
                return this.ToString(null);
            case TypeCode.UInt16:
                return this.ToUInt16(null);
            case TypeCode.UInt32:
                return this.ToUInt32(null);
            case TypeCode.UInt64:
                return this.ToUInt64(null);
            default:
                throw new InvalidCastException(String.Format("Conversion to
{0} is not supported.", conversionType.Name));
        }
    }

    public ushort ToUInt16(IFormatProvider provider)
    {
        if (signBit == SignBit.Negative)
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt16 type.",
```

```
                                                        Int64.Parse(hexString,
NumberStyles.HexNumber)));
      else
         try {
            return Convert.ToUInt16(UInt64.Parse(hexString,
NumberStyles.HexNumber));
         }
         catch (OverflowException e) {
            throw new OverflowException(String.Format("{0} is out of
range of the UInt16 type.", Convert.ToUInt64(hexString, 16)), e);
         }
   }

   public uint ToUInt32(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",
                                                        Int64.Parse(hexString,
NumberStyles.HexNumber)));
      else
         try {
            return Convert.ToUInt32(UInt64.Parse(hexString,
NumberStyles.HexNumber));
         }
         catch (OverflowException) {
            throw new OverflowException(String.Format("{0} is outside the
range of the UInt32 type.",

UInt64.Parse(hexString, NumberStyles.HexNumber)));
         }
   }

   public ulong ToUInt64(IFormatProvider provider)
   {
      if (signBit == SignBit.Negative)
         throw new OverflowException(String.Format("{0} is outside the
range of the UInt64 type.",
                                                        Int64.Parse(hexString,
NumberStyles.HexNumber)));
      else
         return Convert.ToUInt64(hexString, 16);
   }
}
```

The following example shows that a call to the ToUInt64(Object, IFormatProvider) method that passes a `HexString` object as a parameter, in turn, calls the IConvertible.ToUInt64 implementation of the `HexString` class.

```
C#
```

```
public class Example
{
   public static void Main()
   {
      ulong positiveValue = UInt64.MaxValue - 100000;
      long negativeValue = -1;

      HexString positiveString = new HexString();
      positiveString.Sign = (SignBit) Math.Sign((decimal)positiveValue);
      positiveString.Value = positiveValue.ToString("X");

      HexString negativeString = new HexString();
      negativeString.Sign = (SignBit) Math.Sign(negativeValue);
      negativeString.Value = negativeValue.ToString("X");

      try {
         Console.WriteLine("0x{0} converts to {1}.",
positiveString.Value, Convert.ToUInt64(positiveString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt64
type.",
                           Int64.Parse(positiveString.Value,
NumberStyles.HexNumber));
      }

      try {
         Console.WriteLine("0x{0} converts to {1}.",
negativeString.Value, Convert.ToUInt64(negativeString));
      }
      catch (OverflowException) {
         Console.WriteLine("{0} is outside the range of the UInt64
type.",
                           Int64.Parse(negativeString.Value,
NumberStyles.HexNumber));
      }
   }
}
// The example displays the following output:
//       0xFFFFFFFFFFFE795F converts to 18446744073709451615.
//       -1 is outside the range of the UInt64 type.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt64 method of the underlying type of `value`.

`provider` enables the user to specify culture-specific conversion information about the contents of `value`. For example, if `value` is a String that represents a number,

`provider` could supply culture-specific information about the notation used to represent that number.

The base types ignore `provider`; however, the parameter may be used if `value` is a user-defined type that implements the IConvertible interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Single)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified single-precision floating-point number to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

UInt64

`value`, rounded to the nearest 64-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt64.MaxValue.

Examples

The following example attempts to convert each element in an array of Single values to an unsigned long integer. The output shows that fractional values are rounded before the conversion is performed.

C#

```
float[] values= { Single.MinValue, -1.38e10f, -1023.299f, -12.98f,
                  0f, 9.113e-16f, 103.919f, 17834.191f, Single.MaxValue
};
ulong result;

foreach (float value in values)
{
   try {
      result = Convert.ToUInt64(value);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt64 type.",
value);
   }
}
// The example displays the following output:
//    -3.402823E+38 is outside the range of the UInt64 type.
//    -1.38E+10 is outside the range of the UInt64 type.
//    -1023.299 is outside the range of the UInt64 type.
//    -12.98 is outside the range of the UInt64 type.
//    Converted the Single value 0 to the UInt64 value 0.
//    Converted the Single value 9.113E-16 to the UInt64 value 0.
//    Converted the Single value 103.919 to the UInt64 value 104.
```

```
//    Converted the Single value 17834.19 to the UInt64 value 17834.
//    3.402823E+38 is outside the range of the UInt64 type.
```

See also

- Single
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(String, Int32)

> Important
>
> This API is not CLS-compliant.

Converts the string representation of a number in a specified base to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (string? value, int fromBase);
```

Parameters

`value` String

A string that contains the number to convert.

`fromBase` Int32

The base of the number in `value`, which must be 2, 8, 10, or 16.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to the number in `value`, or 0 (zero) if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`fromBase` is not 2, 8, 10, or 16.

-or-

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

ArgumentOutOfRangeException

`value` is Empty.

FormatException

`value` contains a character that is not a valid digit in the base specified by `fromBase`. The exception message indicates that there are no digits to convert if the first character in `value` is invalid; otherwise, the message indicates that `value` contains invalid trailing characters.

OverflowException

`value`, which represents a non-base 10 unsigned number, is prefixed with a negative sign.

-or-

`value` represents a number that is less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example attempts to interpret each element in an array of numeric strings as a hexadecimal value and to convert it to an unsigned long integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] hexStrings = { "8000000000000000", "0FFFFFFFFFFFFFFF",
                              "F000000000000000", "00A3000000000000",
                              "D", "-13", "9AC61", "GAD",
                              "FFFFFFFFFFFFFFFF" };

      foreach (string hexString in hexStrings)
      {
         Console.Write("{0,-18}  -->  ", hexString);
         try {
            ulong number = Convert.ToUInt64(hexString, 16);
            Console.WriteLine("{0,26:N0}", number);
         }
         catch (FormatException) {
            Console.WriteLine("{0,26}", "Bad Format");
         }
         catch (OverflowException) {
            Console.WriteLine("{0,26}", "Numeric Overflow");
         }
         catch (ArgumentException) {
            Console.WriteLine("{0,26}", "Invalid in Base 16");
         }
      }
   }
}
// The example displays the following output:
//    8000000000000000    -->   9,223,372,036,854,775,808
//    0FFFFFFFFFFFFFFF    -->   1,152,921,504,606,846,975
//    F000000000000000    -->  17,293,822,569,102,704,640
//    00A3000000000000    -->      45,880,421,203,836,928
//    D                   -->                          13
//    -13                 -->          Invalid in Base 16
//    9AC61               -->                     633,953
//    GAD                 -->                  Bad Format
//    FFFFFFFFFFFFFFFF    -->            Numeric Overflow
```

Remarks

If `fromBase` is 16, you can prefix the number specified by the `value` parameter with "0x" or "0X".

Because the UInt64 data type supports unsigned values only, the ToUInt64(String, Int32) method assumes that `value` is expressed using unsigned binary representation. In other words, all 64 bits are used to represent the numeric value, and a sign bit is absent. As a result, it is possible to write code in which a signed long integer value that is out of the range of the UInt64 data type is converted to a UInt64 value without the method throwing an exception. The following example converts MinValue to its hexadecimal string representation, and then calls the ToUInt64(String, Int32) method. Instead of throwing an exception, the method displays the message, "0x8000000000000000 converts to 9223372036854775808."

C#

```
// Create a hexadecimal value out of range of the UInt64 type.
string value = Convert.ToString(Int64.MinValue, 16);
// Convert it back to a number.
try
{
   UInt64 number = Convert.ToUInt64(value, 16);
   Console.WriteLine("0x{0} converts to {1}.", value, number);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned long
integer.",
                     value);
}
```

When performing binary operations or numeric conversions, it is always the responsibility of the developer to verify that a method or operator is using the appropriate numeric representation to interpret a particular value. The following example illustrates one technique for ensuring that the method does not inappropriately use binary representation to interpret a value that uses two's complement representation when converting a hexadecimal string to a UInt64 value. The example determines whether a value represents a signed or an unsigned integer while it is converting that value to its string representation. When the example converts the value to a UInt64 value, it checks whether the original value was a signed integer. If so, and if its high-order bit is set (which indicates that the original value was negative), the method throws an exception.

C#

```
// Create a negative hexadecimal value out of range of the UInt64 type.
long sourceNumber = Int64.MinValue;
bool isSigned =
Math.Sign((long)sourceNumber.GetType().GetField("MinValue").GetValue(null
)) == -1;
```

```
string value = Convert.ToString(sourceNumber, 16);
UInt64 targetNumber;
try
{
   targetNumber = Convert.ToUInt64(value, 16);
   if (isSigned && ((targetNumber & 0x8000000000000000) != 0))
      throw new OverflowException();
   else
      Console.WriteLine("0x{0} converts to {1}.", value, targetNumber);
}
catch (OverflowException)
{
   Console.WriteLine("Unable to convert '0x{0}' to an unsigned long
integer.",
                     value);
}
// Displays the following to the console:
//    Unable to convert '0x8000000000000000' to an unsigned long integer.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(UInt64)

> Important
>
> This API is not CLS-compliant.

Returns the specified 64-bit unsigned integer; no actual conversion is performed.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to return.

Returns

UInt64

`value` is returned unchanged.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(SByte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit signed integer to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in a signed byte array to an unsigned long integer.

C#

```
sbyte[] numbers = { SByte.MinValue, -1, 0, 10, SByte.MaxValue };
ulong result;

foreach (sbyte number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
                        number.GetType().Name, number);
```

```
    }
}
// The example displays the following output:
//    The SByte value -128 is outside the range of the UInt64 type.
//    The SByte value -1 is outside the range of the UInt64 type.
//    Converted the SByte value 0 to the UInt64 value 0.
//    Converted the SByte value 10 to the UInt64 value 10.
//    Converted the SByte value 127 to the UInt64 value 127.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Int16)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 16-bit signed integer to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in an array of 16-bit integers to an unsigned long integer.

C#

```
short[] numbers= { Int16.MinValue, -1, 0, 121, 340, Int16.MaxValue };
ulong result;

foreach (short number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to a {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt64 type.",
number);
   }
}
// The example displays the following output:
//    -32768 is outside the range of the UInt64 type.
//    -1 is outside the range of the UInt64 type.
//    Converted the Int16 value 0 to a UInt64 value 0.
//    Converted the Int16 value 121 to a UInt64 value 121.
//    Converted the Int16 value 340 to a UInt64 value 340.
//    Converted the Int16 value 32767 to a UInt64 value 32767.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Int64)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 64-bit signed integer to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in an array of long integers to an unsigned long integer.

C#

```
long[] numbers = { Int64.MinValue, -19432, -18, 0, 121, 340,
Int64.MaxValue };
ulong result;
foreach (long number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int64 value -9223372036854775808 is outside the range of the
UInt64 type.
//    The Int64 value -19432 is outside the range of the UInt64 type.
//    The Int64 value -18 is outside the range of the UInt64 type.
//    Converted the Int64 value 0 to the UInt64 value 0.
//    Converted the Int64 value 121 to the UInt64 value 121.
//    Converted the Int64 value 340 to the UInt64 value 340.
//    Converted the Int64 value 9223372036854775807 to a UInt64 value
9223372036854775807.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Int32)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 32-bit signed integer to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero.

Examples

The following example attempts to convert each element in an array of integers to an unsigned long integer.

C#

```
int[] numbers = { Int32.MinValue, -1, 0, 121, 340, Int32.MaxValue };
ulong result;

foreach (int number in numbers)
{
   try {
      result = Convert.ToUInt64(number);
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        number.GetType().Name, number,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
                        number.GetType().Name, number);
   }
}
// The example displays the following output:
//    The Int32 value -2147483648 is outside the range of the UInt64
type.
//    The Int32 value -1 is outside the range of the UInt64 type.
//    Converted the Int32 value 0 to the UInt64 value 0.
//    Converted the Int32 value 121 to the UInt64 value 121.
//    Converted the Int32 value 340 to the UInt64 value 340.
//    Converted the Int32 value 2147483647 to the UInt64 value
2147483647.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToUInt64(Double)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified double-precision floating-point number to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

UInt64

`value`, rounded to the nearest 64-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt64.MaxValue.

Examples

The following example attempts to convert each element in an array of Double values to an unsigned long integer.

C#

```
double[] values= { Double.MinValue, -1.38e10, -1023.299, -12.98,
                   0, 9.113e-16, 103.919, 17834.191, Double.MaxValue };
ulong result;

foreach (double value in values)
{
   try {
      result = Convert.ToUInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt64 type.",
value);
   }
}
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the UInt64 type.
//    -13800000000 is outside the range of the UInt64 type.
//    -1023.299 is outside the range of the UInt64 type.
//    -12.98 is outside the range of the UInt64 type.
//    Converted the Double value '0' to the UInt64 value 0.
//    Converted the Double value '9.113E-16' to the UInt64 value 0.
//    Converted the Double value '103.919' to the UInt64 value 104.
//    Converted the Double value '17834.191' to the UInt64 value 17834.
//    1.79769313486232E+308 is outside the range of the UInt64 type.
```

See also

- Double
- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Decimal)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified decimal number to an equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

UInt64

`value`, rounded to the nearest 64-bit unsigned integer. If `value` is halfway between two whole numbers, the even number is returned; that is, 4.5 is converted to 4, and 5.5 is converted to 6.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than zero or greater than UInt64.MaxValue.

Examples

The following example attempts to convert each element in an array of Decimal values to an unsigned long integer.

C#

```
decimal[] values= { Decimal.MinValue, -1034.23m, -12m, 0m, 147m,
                    199.55m, 9214.16m, Decimal.MaxValue };
ulong result;

foreach (decimal value in values)
{
   try {
      result = Convert.ToUInt64(value);
      Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("{0} is outside the range of the UInt64 type.",
                        value);
   }
}
// The example displays the following output:
//    -79228162514264337593543950335 is outside the range of the UInt64
type.
//    -1034.23 is outside the range of the UInt64 type.
//    -12 is outside the range of the UInt64 type.
//    Converted the Decimal value '0' to the UInt64 value 0.
//    Converted the Decimal value '147' to the UInt64 value 147.
//    Converted the Decimal value '199.55' to the UInt64 value 200.
//    Converted the Decimal value '9214.16' to the UInt64 value 9214.
//    79228162514264337593543950335 is outside the range of the UInt64
type.
```

See also

- Round

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(DateTime)

> Important
>
> This API is not CLS-compliant.

Calling this method always throws InvalidCastException.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (DateTime value);
```

Parameters

`value` DateTime

The date and time value to convert.

Returns

UInt64

This conversion is not supported. No value is returned.

Attributes CLSCompliantAttribute

Exceptions

InvalidCastException

This conversion is not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToUInt64(Char)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified Unicode character to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a character array to an unsigned long integer.

C#

```
char[] chars = { 'a', 'z', '\u0007', '\u03FF',
                 '\u7FFF', '\uFFFE' };
ulong result;

foreach (char ch in chars)
{
   result = Convert.ToUInt64(ch);
   Console.WriteLine("Converted the {0} value '{1}' to the {2} value
{3}.",
                     ch.GetType().Name, ch,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Char value 'a' to the UInt64 value 97.
//    Converted the Char value 'z' to the UInt64 value 122.
//    Converted the Char value '' to the UInt64 value 7.
//    Converted the Char value '?' to the UInt64 value 1023.
//    Converted the Char value '?' to the UInt64 value 32767.
//    Converted the Char value '?' to the UInt64 value 65534.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Byte)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified 8-bit unsigned integer to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (byte value);
```

Parameters

value Byte

The 8-bit unsigned integer to convert.

Returns

UInt64

A 64-bit signed integer that is equivalent to `value`.

Attributes CLSCompliantAttribute

Examples

The following example converts each element in a byte array to an unsigned long integer.

C#

```
byte[] bytes = { Byte.MinValue, 14, 122, Byte.MaxValue};
ulong result;

foreach (byte byteValue in bytes)
{
   result = Convert.ToUInt64(byteValue);
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     byteValue.GetType().Name, byteValue,
                     result.GetType().Name, result);
}
// The example displays the following output:
//    Converted the Byte value 0 to the UInt64 value 0.
//    Converted the Byte value 14 to the UInt64 value 14.
//    Converted the Byte value 122 to the UInt64 value 122.
//    Converted the Byte value 255 to the UInt64 value 255.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Boolean)

> Important
>
> This API is not CLS-compliant.

Converts the specified Boolean value to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (bool value);
```

Parameters

`value` Boolean

The Boolean value to convert.

Returns

UInt64

The number 1 if `value` is `true`; otherwise, 0.

Attributes CLSCompliantAttribute

Examples

The following example converts Boolean `true` and `false` values to unsigned long integers.

C#

```
bool falseFlag = false;
bool trueFlag = true;

Console.WriteLine("{0} converts to {1}.", falseFlag,
                  Convert.ToUInt64(falseFlag));
Console.WriteLine("{0} converts to {1}.", trueFlag,
                  Convert.ToUInt64(trueFlag));
// The example displays the following output:
//       False converts to 0.
//       True converts to 1.
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUInt64(Object)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified object to a 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (object? value);
```

Parameters

`value` Object

An object that implements the IConvertible interface, or `null`.

Returns

UInt64

A 64-bit unsigned integer that is equivalent to `value`, or zero if `value` is `null`.

Attributes CLSCompliantAttribute

Exceptions

FormatException

`value` is not in an appropriate format.

InvalidCastException

`value` does not implement the IConvertible interface.

-or-

The conversion is not supported.

OverflowException

`value` represents a number that is less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example attempts to convert each element in an object array to an unsigned long integer.

C#

```
object[] values = { true, -12, 163, 935, 'x', new DateTime(2009, 5, 12),
                    "104", "103.0", "-1",
                    "1.00e2", "One", 1.00e2, 16.3e42};
ulong result;

foreach (object value in values)
{
   try {
      result = Convert.ToUInt64(value);
```

```
      Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                        value.GetType().Name, value,
                        result.GetType().Name, result);
   }
   catch (OverflowException) {
      Console.WriteLine("The {0} value {1} is outside the range of the
UInt64 type.",
                        value.GetType().Name, value);
   }
   catch (FormatException) {
      Console.WriteLine("The {0} value {1} is not in a recognizable
format.",
                        value.GetType().Name, value);
   }
   catch (InvalidCastException) {
      Console.WriteLine("No conversion to a UInt64 exists for the {0}
value {1}.",
                        value.GetType().Name, value);
   }
}
// The example displays the following output:
//    Converted the Boolean value True to the UInt64 value 1.
//    The Int32 value -12 is outside the range of the UInt64 type.
//    Converted the Int32 value 163 to the UInt64 value 163.
//    Converted the Int32 value 935 to the UInt64 value 935.
//    Converted the Char value x to the UInt64 value 120.
//    No conversion to a UInt64 exists for the DateTime value 5/12/2009
12:00:00 AM.
//    Converted the String value 104 to the UInt64 value 104.
//    The String value 103.0 is not in a recognizable format.
//    The String value -1 is outside the range of the UInt64 type.
//    The String value 1.00e2 is not in a recognizable format.
//    The String value One is not in a recognizable format.
//    Converted the Double value 100 to the UInt64 value 100.
//    The Double value 1.63E+43 is outside the range of the UInt64 type.
```

Remarks

The return value is the result of invoking the IConvertible.ToUInt64 method of the underlying type of `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.TryFromBase64Chars Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the specified span containing a string representation that is encoded with base-64 digits into a span of 8-bit unsigned integers.

C#

```
public static bool TryFromBase64Chars (ReadOnlySpan<char> chars, Span<byte>
bytes, out int bytesWritten);
```

Parameters

chars ReadOnlySpan<Char>

A span containing the string representation that is encoded with base-64 digits.

bytes Span<Byte>

The span in which to write the converted 8-bit unsigned integers. If this method returns `false`, either the span remains unmodified or contains an incomplete conversion of `chars`, up to the last valid character.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `bytes`.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Convert.TryFromBase64String(String, Span<Byte>, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the specified string representation that is encoded with base-64 digits into a span of 8-bit unsigned integers.

C#

```
public static bool TryFromBase64String (string s, Span<byte> bytes, out int
bytesWritten);
```

Parameters

s String

The string representation that is encoded with base-64 digits.

bytes Span<Byte>

The span in which to write the converted 8-bit unsigned integers. When this method returns `false`, either the span remains unmodified or contains an incomplete conversion of `s`, up to the last valid character.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `bytes`.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Exceptions

ArgumentNullException

`s` is `null`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Convert.TryToBase64Chars Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the 8-bit unsigned integers inside the specified read-only span into their equivalent string representation that is encoded with base-64 digits. You can optionally specify whether to insert line breaks in the return value.

C#

```
public static bool TryToBase64Chars (ReadOnlySpan<byte> bytes, Span<char>
chars, out int charsWritten, Base64FormattingOptions options =
System.Base64FormattingOptions.None);
```

Parameters

bytes ReadOnlySpan<Byte>

A read-only span of 8-bit unsigned integers.

chars Span<Char>

The span in which to write the string representation in base 64 of the elements in `bytes`. If the length of `bytes` is 0, or when this method returns `false`, nothing is written into this parameter.

charsWritten Int32

When this method returns, contains the total number of characters written into `chars`.

options Base64FormattingOptions

One of the enumeration values that specify whether to insert line breaks in the return value. The default value is None.

Returns

Boolean

`true` if the conversion is successful; otherwise, `false`.

Exceptions

ArgumentException

`options` is not a valid Base64FormattingOptions value.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Converter<TInput,TOutput> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a method that converts an object from one type to another type.

C#

```
public delegate TOutput Converter<in TInput,out TOutput>(TInput input);
```

Type Parameters

`TInput`

The type of object that is to be converted.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TOutput`

The type the input object is to be converted to.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`input` TInput

The object to convert.

Return Value

TOutput

The `TOutput` that represents the converted `TInput`.

Examples

This section contains two code examples. The first demonstrates the Converter<TInput,TOutput> delegate with the ConvertAll method of the Array class, and the second demonstrates the delegate with the ConvertAll method of the List<T> generic class.

Example 1

The following code example defines a method named `PointFToPoint` that converts a PointF structure to a Point structure. The example then creates an array of PointF structures, creates a `Converter<PointF, Point>` delegate (`Converter(Of PointF, Point)` in Visual Basic) to represent the `PointFToPoint` method, and passes the delegate to the ConvertAll method. The ConvertAll method passes each element of the input list to the `PointFToPoint` method and puts the converted elements into a new list of Point structures. Both lists are displayed.

C#

```
using System;
using System.Drawing;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        // Create an array of PointF objects.
        PointF[] apf = {
            new PointF(27.8F, 32.62F),
            new PointF(99.3F, 147.273F),
            new PointF(7.5F, 1412.2F) };

        // Display each element in the PointF array.
        Console.WriteLine();
        foreach( PointF p in apf )
            Console.WriteLine(p);

        // Convert each PointF element to a Point object.
        Point[] ap = Array.ConvertAll(apf,
            new Converter<PointF, Point>(PointFToPoint));

        // Display each element in the Point array.
        Console.WriteLine();
        foreach( Point p in ap )
        {
            Console.WriteLine(p);
        }
    }
```

```
    public static Point PointFToPoint(PointF pf)
    {
        return new Point(((int) pf.X), ((int) pf.Y));
    }
}

/* This code example produces the following output:

{X=27.8, Y=32.62}
{X=99.3, Y=147.273}
{X=7.5, Y=1412.2}

{X=27,Y=32}
{X=99,Y=147}
{X=7,Y=1412}
 */
```

Example 2

The following code example defines a method named `PointFToPoint` that converts a PointF structure to a Point structure. The example then creates a List<T> of PointF structures, creates a `Converter\<PointF, Point>` delegate (`Converter(Of PointF, Point)` in Visual Basic) to represent the `PointFToPoint` method, and passes the delegate to the ConvertAll method. The ConvertAll method passes each element of the input list to the `PointFToPoint` method and puts the converted elements into a new list of Point structures. Both lists are displayed.

C#

```
using System;
using System.Drawing;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        List<PointF> lpf = new List<PointF>();

        lpf.Add(new PointF(27.8F, 32.62F));
        lpf.Add(new PointF(99.3F, 147.273F));
        lpf.Add(new PointF(7.5F, 1412.2F));

        Console.WriteLine();
        foreach( PointF p in lpf )
        {
            Console.WriteLine(p);
        }
```

```
        List<Point> lp = lpf.ConvertAll(
            new Converter<PointF, Point>(PointFToPoint));

        Console.WriteLine();
        foreach( Point p in lp )
        {
            Console.WriteLine(p);
        }
    }

    public static Point PointFToPoint(PointF pf)
    {
        return new Point(((int) pf.X), ((int) pf.Y));
    }
}

/* This code example produces the following output:

{X=27.8, Y=32.62}
{X=99.3, Y=147.273}
{X=7.5, Y=1412.2}

{X=27,Y=32}
{X=99,Y=147}
{X=7,Y=1412}
 */
```

Remarks

This delegate is used by the ConvertAll method of the Array class and the ConvertAll method of the List<T> class to convert each element of the collection from one type to another.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DataMisalignedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.InteropServices.dll

The exception that is thrown when a unit of data is read from or written to an address that is not a multiple of the data size. This class cannot be inherited.

C#

```
public sealed class DataMisalignedException : SystemException
```

Inheritance Object → Exception → SystemException → DataMisalignedException

Remarks

Some microprocessors, such as 64-bit processors, can issue hardware exceptions when data is read or written to a misaligned address (that is, when an address is not a multiple of the size of the data being accessed). DataMisalignedException is typically thrown when 16-bit, 32-bit, or 64-bit data is read from or written to an address that is not a multiple of two, four, or eight bytes.

The DataMisalignedException exposes the underlying hardware exception, which allows you to respond in an appropriate manner. For example, a routine in your application attempts to copy a sequence of bytes from one location to another in 64-bit units. However, the write operation starts at a misaligned address, which causes the common language runtime to throw a DataMisalignedException. Your routine responds to the exception by copying the data one byte at a time.

Constructors

Expand table

DataMisalignedException()	Initializes a new instance of the DataMisalignedException class.

DataMisaligned Exception(String)	Initializes a new instance of the DataMisalignedException class using the specified error message.
DataMisaligned Exception(String, Exception)	Initializes a new instance of the DataMisalignedException class using the specified error message and underlying exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.

	(Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DataMisalignedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.InteropServices.dll

Initializes a new instance of the DataMisalignedException class.

Overloads

Expand table

DataMisalignedException()	Initializes a new instance of the DataMisalignedException class.
DataMisalignedException(String)	Initializes a new instance of the DataMisalignedException class using the specified error message.
DataMisalignedException(String, Exception)	Initializes a new instance of the DataMisalignedException class using the specified error message and underlying exception.

DataMisalignedException()

Initializes a new instance of the DataMisalignedException class.

C#

```
public DataMisalignedException ();
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DataMisalignedException(String)

Initializes a new instance of the DataMisalignedException class using the specified error message.

C#

```
public DataMisalignedException (string? message);
```

Parameters

`message` String

A String object that describes the error. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DataMisalignedException(String, Exception)

Initializes a new instance of the DataMisalignedException class using the specified error message and underlying exception.

```
C#

public DataMisalignedException (string? message, Exception?
innerException);
```

Parameters

`message` String

A String object that describes the error. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

`innerException` Exception

The exception that is the cause of the current DataMisalignedException. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateOnly Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents dates with values ranging from January 1, 0001 Anno Domini (Common Era) through December 31, 9999 A.D. (C.E.) in the Gregorian calendar.

C#

```
public readonly struct DateOnly : IComparable, IComparable<DateOnly>,
IEquatable<DateOnly>, IParsable<DateOnly>, ISpanFormattable,
ISpanParsable<DateOnly>, IUtf8SpanFormattable
```

Inheritance Object → ValueType → DateOnly

Implements IComparable , IComparable<DateOnly> , IEquatable<DateOnly> , IFormattable , ISpanFormattable , IParsable<DateOnly> , IParsable<TSelf> , ISpanParsable<DateOnly> , IUtf8SpanFormattable

Constructors

Expand table

DateOnly(Int32, Int32, Int32)	Creates a new instance of the DateOnly structure to the specified year, month, and day.
DateOnly(Int32, Int32, Int32, Calendar)	Creates a new instance of the DateOnly structure to the specified year, month, and day for the specified calendar.

Properties

Expand table

Day	Gets the day component of the date represented by this instance.

Day Number	Gets the number of days since January 1, 0001 in the Proleptic Gregorian calendar represented by this instance.
Day OfWeek	Gets the day of the week represented by this instance.
DayOfYear	Gets the day of the year represented by this instance.
MaxValue	Gets the latest possible date that can be created.
MinValue	Gets the earliest possible date that can be created.
Month	Gets the month component of the date represented by this instance.
Year	Gets the year component of the date represented by this instance.

Methods

Expand table

AddDays(Int32)	Adds the specified number of days to the value of this instance.
AddMonths(Int32)	Adds the specified number of months to the value of this instance.
AddYears(Int32)	Adds the specified number of years to the value of this instance.
CompareTo(DateOnly)	Compares the value of this instance to a specified DateOnly value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.
CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified DateOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.
Deconstruct(Int32, Int32, Int32)	Deconstructs DateOnly by Year, Month, and Day.
Equals(DateOnly)	Returns a value indicating whether the value of this instance is equal to the value of the specified DateOnly instance.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

FromDateTime(DateTime)	Returns a DateOnly instance that is set to the date part of the specified `dateTime`.
FromDayNumber(Int32)	Creates a new instance of the DateOnly structure to the specified number of days.
GetHashCode()	Returns the hash code for this instance.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.
Parse(String)	Converts a string that contains string representation of a date to its DateOnly equivalent by using the conventions of the current culture.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(String, IFormatProvider, DateTimeStyles)	Converts a string that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[])	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format, culture-

	specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String[])	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ToDateTime(TimeOnly)	Returns a DateTime that is set to the date of this DateOnly instance and the time of specified input time.
ToDateTime(TimeOnly, DateTimeKind)	Returns a DateTime instance with the specified input kind that is set to the date of this DateOnly instance and the time of specified input time.
ToLongDateString()	Converts the value of the current DateOnly object to its equivalent long date string representation.
ToShortDateString()	Converts the value of the current DateOnly object to its equivalent short date string representation.
ToString()	Converts the value of the current DateOnly object to its equivalent string representation using the formatting conventions of the current culture. The DateOnly object will be formatted in short form.
ToString(IFormatProvider)	Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the value of the current DateOnly object to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString(String, IFormatProvider)	Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current DateOnly instance into the provided span of characters.

TryParse(ReadOnlySpan<Char>, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateOnly)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.
TryParse(String, DateOnly)	Converts the specified string representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateOnly)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnly Span<Char>, ReadOnly Span<Char>, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnly Span<Char>, ReadOnly Span<Char>, IFormatProvider, Date TimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnlyequivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnly Span<Char>, String[], DateOnly)	Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnly Span<Char>, String[], IFormat Provider, DateTimeStyles, DateOnly)	Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, Date Only)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], DateOnly)	Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

Operators

Expand table

Equality(DateOnly, DateOnly)	Determines whether two specified instances of DateOnly are equal.
GreaterThan(DateOnly, DateOnly)	Determines whether one specified DateOnly is later than another specified DateTime.
GreaterThanOrEqual(DateOnly, DateOnly)	Determines whether one specified DateOnly represents a date that is the same as or later than another specified DateOnly.
Inequality(DateOnly, DateOnly)	Determines whether two specified instances of DateOnly are not equal.
LessThan(DateOnly, DateOnly)	Determines whether one specified DateOnly is earlier than another specified DateOnly.
LessThanOrEqual(DateOnly, DateOnly)	Determines whether one specified DateOnly represents a date that is the same as or earlier than another specified DateOnly.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

DateOnly(Int32, Int32, Int32)	Creates a new instance of the DateOnly structure to the specified year, month, and day.
DateOnly(Int32, Int32, Int32, Calendar)	Creates a new instance of the DateOnly structure to the specified year, month, and day for the specified calendar.

DateOnly(Int32, Int32, Int32)

Creates a new instance of the DateOnly structure to the specified year, month, and day.

C#

```
public DateOnly (int year, int month, int day);
```

Parameters

year Int32

The year (1 through 9999).

month Int32

The month (1 through 12).

day Int32

The day (1 through the number of days in `month`).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DateOnly(Int32, Int32, Int32, Calendar)

Creates a new instance of the DateOnly structure to the specified year, month, and day for the specified calendar.

C#

```
public DateOnly (int year, int month, int day,
System.Globalization.Calendar calendar);
```

Parameters

`year` Int32

The year (1 through the number of years in calendar).

`month` Int32

The month (1 through the number of months in calendar).

`day` Int32

The day (1 through the number of days in `month`).

`calendar` Calendar

The calendar that is used to interpret year, month, and day. `month`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.Day Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day component of the date represented by this instance.

C#

```
public int Day { get; }
```

Property Value

Int32

A number representing the day component of the date represented by this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.DayNumber Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of days since January 1, 0001 in the Proleptic Gregorian calendar represented by this instance.

C#

```
public int DayNumber { get; }
```

Property Value

Int32

The number of days since January 1, 0001 in the Proleptic Gregorian calendar represented by this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.DayOfWeek Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the week represented by this instance.

C#

```
public DayOfWeek DayOfWeek { get; }
```

Property Value

DayOfWeek

A number that represents the day of the week represented by this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.DayOfYear Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the year represented by this instance.

C#

```
public int DayOfYear { get; }
```

Property Value

Int32

A number that represents the day of the year represented by this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the latest possible date that can be created.

C#

```
public static DateOnly MaxValue { get; }
```

Property Value

DateOnly

A DateOnly representing the latest possible date that can be created.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the earliest possible date that can be created.

C#

```
public static DateOnly MinValue { get; }
```

Property Value

DateOnly

A DateOnly representing the earliest possible date that can be created.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.Month Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the month component of the date represented by this instance.

C#

```
public int Month { get; }
```

Property Value

Int32

A number that represents the month component of the date.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.Year Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the year component of the date represented by this instance.

C#

```
public int Year { get; }
```

Property Value

Int32

A number that represents the year component of the date.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.AddDays(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds the specified number of days to the value of this instance.

C#

```
public DateOnly AddDays (int value);
```

Parameters

`value` Int32

The number of days to add. To subtract days, specify a negative number.

Returns

DateOnly

An instance whose value is the sum of the date represented by this instance and the number of days represented by value.

Exceptions

ArgumentOutOfRangeException

The resulting value is greater than DateOnly.MaxValue.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.AddMonths(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds the specified number of months to the value of this instance.

```
C#
public DateOnly AddMonths (int value);
```

Parameters

`value` Int32

A number of months. The months parameter can be negative or positive.

Returns

DateOnly

An object whose value is the sum of the date represented by this instance and months.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateOnly.AddYears(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds the specified number of years to the value of this instance.

```
C#
public DateOnly AddYears (int value);
```

Parameters

`value` Int32

A number of years. The value parameter can be negative or positive.

Returns

DateOnly

An object whose value is the sum of the date represented by this instance and the number of years represented by value.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateOnly.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(DateOnly)	Compares the value of this instance to a specified DateOnly value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.
CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified DateOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.

CompareTo(DateOnly)

Compares the value of this instance to a specified DateOnly value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.

C#

```
public int CompareTo (DateOnly value);
```

Parameters

`value` DateOnly

The object to compare to the current instance.

Returns

Int32

Less than zero if this instance is earlier than value. Greater than zero if this instance is later than value. Zero if this instance is the same as value.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

CompareTo(Object)

Compares the value of this instance to a specified object that contains a specified DateOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateOnly value.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

A boxed object to compare, or `null`.

Returns

Int32

Less than zero if this instance is earlier than value. Greater than zero if this instance is later than value. Zero if this instance is the same as value.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.Deconstruct(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Deconstructs DateOnly by Year, Month, and Day.

C#

```
public void Deconstruct (out int year, out int month, out int day);
```

Parameters

year Int32

When this method returns, represents the Year value of this DateOnly instance.

month Int32

When this method returns, represents the Month value of this DateOnly instance.

day Int32

When this method returns, represents the Day value of this DateOnly instance.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateOnly.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(DateOnly)	Returns a value indicating whether the value of this instance is equal to the value of the specified DateOnly instance.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(DateOnly)

Returns a value indicating whether the value of this instance is equal to the value of the specified DateOnly instance.

```
C#

public bool Equals (DateOnly value);
```

Parameters

`value` DateOnly

The object to compare to this instance.

Returns

Boolean

`true` if the value parameter equals the value of this instance; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

```
C#
public override bool Equals (object? value);
```

Parameters

`value` Object

The object to compare to this instance.

Returns

Boolean

`true` if value is an instance of DateOnly and equals the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.FromDateTime(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a DateOnly instance that is set to the date part of the specified `dateTime`.

```
C#
public static DateOnly FromDateTime (DateTime dateTime);
```

Parameters

`dateTime` DateTime

The DateTime instance.

Returns

DateOnly

The DateOnly instance composed of the date part of the specified input time `dateTime` instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateOnly.FromDayNumber(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new instance of the DateOnly structure to the specified number of days.

C#

```
public static DateOnly FromDayNumber (int dayNumber);
```

Parameters

dayNumber Int32

The number of days since January 1, 0001 in the Proleptic Gregorian calendar.

Returns

DateOnly

A DateOnly structure instance to the specified number of days.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateOnly.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String)	Converts a string that contains string representation of a date to its DateOnly equivalent by using the conventions of the current culture.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.
Parse(String, IFormatProvider, DateTimeStyles)	Converts a string that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.

Parse(String)

Converts a string that contains string representation of a date to its DateOnly equivalent by using the conventions of the current culture.

C#

```
public static DateOnly Parse (string s);
```

Parameters

`s` String

The string that contains the string to parse.

Returns

DateOnly

An object that is equivalent to the date contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#

public static DateOnly Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

DateOnly

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, IFormatProvider)

Parses a string into a value.

```
C#

public static DateOnly Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

DateOnly

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts a memory span that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.

C#

```
public static DateOnly Parse (ReadOnlySpan<char> s, IFormatProvider?
provider = default, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

The memory span that contains the string to parse.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by provider and styles.

Exceptions

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(String, IFormatProvider, DateTimeStyles)

Converts a string that contains string representation of a date to its DateOnly equivalent by using culture-specific format information and a formatting style.

C#

```
public static DateOnly Parse (string s, IFormatProvider? provider,
System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` String

The string that contains the string to parse.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s` for the parse operation to succeed, and that defines how to interpret the parsed date. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by provider and styles.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[])	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String[])	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

ParseExact(String, String)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

ParseExact(String, String[], IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static DateOnly ParseExact (string s, string[] formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s String

A string containing the characters that represent a date to convert.

formats String[]

An array of allowable formats of `s`.

provider IFormatProvider

An object that supplies culture-specific formatting information about `s`.

style DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String, IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static DateOnly ParseExact (string s, string format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s String

A string containing the characters that represent a date to convert.

`format` String

A string containing the characters that represent a format specifier that defines the required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that provides additional information about `s`, about style elements that may be present in `s`, or about the conversion from `s` to a DateOnly value. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)

Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. The

format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

```
C#

public static DateOnly ParseExact (ReadOnlySpan<char> s, string[]
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style = System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent a date to convert.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, String[])

Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

```
C#

public static DateOnly ParseExact (ReadOnlySpan<char> s, string[]
formats);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters that represent a date to convert.

formats String[]

An array of allowable formats of `s`.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Exceptions

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String[])

Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats. The format of the string representation must match at

least one of the specified formats exactly or an exception is thrown.

```C#
public static DateOnly ParseExact (string s, string[] formats);
```

Parameters

`s` String

A span containing the characters that represent a date to convert.

`formats` String[]

An array of allowable formats of `s`.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String)

Converts the specified string representation of a date to its DateOnly equivalent using the specified format. The format of the string representation must match the

specified format exactly or an exception is thrown.

```C#
public static DateOnly ParseExact (string s, string format);
```

Parameters

`s` String

A string containing the characters that represent a date to convert.

`format` String

A string that represent a format specifier that defines the required format of `s`.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date.

Remarks

The ParseExact(String, String) method parses the string representation of a date, which must be in the format defined by the `format` parameter. It also requires that the <Date> element of the string representation of a date appear in the order specified by `format`, and that `s` have no white space other than that permitted by `format`. The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of `s`. For details about valid formatting codes, see Standard Date and Time Format Strings or Custom Date and Time Format Strings.

Note

If `format` is a custom format pattern that does not include date separators (such as "yyyyMMdd"), use the widest form of each custom format specifier. For example, if you want to specify months in the format pattern, specify the wider form, "MM", instead of the narrower form, "M".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static DateOnly ParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider = default,
System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters that represent a date to convert.

format ReadOnlySpan<Char>

A span containing the characters that represent a format specifier that defines the required format of `s`.

provider IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateOnly

An object that is equivalent to the date contained in `s`, as specified by format, provider, and style.

Exceptions

FormatException

`s` does not contain a valid string representation of a date.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.ToDateTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToDateTime(TimeOnly)	Returns a DateTime that is set to the date of this DateOnly instance and the time of specified input time.
ToDateTime(TimeOnly, DateTimeKind)	Returns a DateTime instance with the specified input kind that is set to the date of this DateOnly instance and the time of specified input time.

ToDateTime(TimeOnly)

Returns a DateTime that is set to the date of this DateOnly instance and the time of specified input time.

C#

```
public DateTime ToDateTime (TimeOnly time);
```

Parameters

`time` TimeOnly

The time of the day.

Returns

DateTime

The DateTime instance composed of the date of the current DateOnly instance and the time specified by the input time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToDateTime(TimeOnly, DateTimeKind)

Returns a DateTime instance with the specified input kind that is set to the date of this DateOnly instance and the time of specified input time.

C#

```
public DateTime ToDateTime (TimeOnly time, DateTimeKind kind);
```

Parameters

`time` TimeOnly

The time of the day.

`kind` DateTimeKind

One of the enumeration values that indicates whether ticks specifies a local time, Coordinated Universal Time (UTC), or neither.

Returns

DateTime

The DateTime instance composed of the date of the current DateOnly instance and the time specified by the input time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.ToLongDateString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateOnly object to its equivalent long date string representation.

C#

```
public string ToLongDateString ();
```

Returns

String

A string that contains the long date string representation of the current DateOnly object.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.ToShortDateString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateOnly object to its equivalent short date string representation.

C#

```
public string ToShortDateString ();
```

Returns

String

A string that contains the short date string representation of the current DateOnly object.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString()	Converts the value of the current DateOnly object to its equivalent string representation using the formatting conventions of the current culture. The DateOnly object will be formatted in short form.
ToString(IFormatProvider)	Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the value of the current DateOnly object to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString(String, IFormatProvider)	Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.

ToString()

Converts the value of the current DateOnly object to its equivalent string representation using the formatting conventions of the current culture. The DateOnly object will be formatted in short form.

C#

```
public override string ToString ();
```

Returns

String

A string that contains the short date string representation of the current DateOnly object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString(IFormatProvider)

Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of value of the current DateOnly object as specified by provider.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString(String)

Converts the value of the current DateOnly object to its equivalent string representation using the specified format and the formatting conventions of the current culture.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom date format string.

Returns

String

A string representation of value of the current DateOnly object as specified by format.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString(String, IFormatProvider)

Converts the value of the current DateOnly object to its equivalent string representation using the specified culture-specific format information.

```
C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A standard or custom date format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of value of the current DateOnly object as specified by format and provider.

Implements

ToString(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current DateOnly instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current DateOnly instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateOnly.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(String, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateOnly)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, DateOnly)	Converts the specified string representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateOnly)	Tries to parse a span of characters into a value.

TryParse(String, IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified string representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider,
System.Globalization.DateTimeStyles style, out DateOnly result);
```

Parameters

`s` String

A string containing the characters that represent a date to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified span representation of a date to its DateOnly equivalent using the specified array of formats, culture-specific format information, and style. And returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, System.Globalization.DateTimeStyles style, out DateOnly
result);
```

Parameters

`s` ReadOnlySpan<Char>

A string containing the characters that represent a date to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(String, IFormatProvider, DateOnly)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
DateOnly result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateOnly

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, DateOnly)

Converts the specified span representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out DateOnly result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing the date to convert.

result DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(String, DateOnly)

Converts the specified string representation of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (string? s, out DateOnly result);
```

Parameters

`s` String

A string containing the characters representing the date to convert.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateOnly)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out DateOnly result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateOnly

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateOnly.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], DateOnly)	Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, DateOnly)	Converts the specified string representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], DateOnly)	Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnlyequivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateOnly)	Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string?[]? formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
DateOnly result);
```

Parameters

`s` String

The string containing the date to parse.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date. A typical value to specify is None.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, DateOnly)

Converts the specified span representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, out DateOnly result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing a date to convert.

format ReadOnlySpan<Char>

The required format of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` is an empty string, or does not contain a date that correspond to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, String[], DateOnly)

Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s, string?[]?
formats, out DateOnly result);
```

Parameters

`s` ReadOnlySpan<Char>

The span containing the string to parse.

`formats` String[]

An array of allowable formats of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is an empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String, DateOnly)

Converts the specified string representation of a date to its DateOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string? format, out DateOnly
result);
```

Parameters

`s` String

A string containing the characters representing a date to convert.

`format` String

The required format of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string, or does not contain a date that correspond to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String[], DateOnly)

Converts the specified string of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string?[]? formats, out
DateOnly result);
```

Parameters

`s` String

The string containing date to parse.

`formats` String[]

An array of allowable formats of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion

failed. The conversion fails if `s` is an empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified span representation of a date to its DateOnlyequivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider,
System.Globalization.DateTimeStyles style, out DateOnly result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing a date to convert.

format ReadOnlySpan<Char>

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if the `s` is an empty string, or does not contain a date that correspond to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified char span of a date to its DateOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s, string?[]?
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style, out DateOnly result);
```

Parameters

`s` ReadOnlySpan<Char>

The span containing the string to parse.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date. A typical value to specify is None.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string, or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateOnly)

Converts the specified span representation of a date to its DateOnly equivalent using the specified format, culture-specific format information, and style. The format of the

string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

```C#
public static bool TryParseExact (string? s, string? format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
DateOnly result);
```

Parameters

`s` String

A span containing the characters representing a date to convert.

`format` String

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` DateOnly

When this method returns, contains the DateOnly value equivalent to the date contained in `s`, if the conversion succeeded, or DateOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string, or does not contain a date that correspond to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.Equality(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of DateOnly are equal.

```
C#
public static bool operator == (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left and right represent the same date; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.GreaterThan(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateOnly is later than another specified DateTime.

```
C#
public static bool operator > (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left is later than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.GreaterThanOrEqual(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateOnly represents a date that is the same as or later than another specified DateOnly.

C#

```
public static bool operator >= (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left is the same as or later than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.Inequality(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of DateOnly are not equal.

```
C#
public static bool operator != (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left and right do not represent the same date; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.LessThan(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateOnly is earlier than another specified DateOnly.

```
C#
public static bool operator < (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left is earlier than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateOnly.LessThanOrEqual(DateOnly, DateOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateOnly represents a date that is the same as or earlier than another specified DateOnly.

```
C#
public static bool operator <= (DateOnly left, DateOnly right);
```

Parameters

`left` DateOnly

The first object to compare.

`right` DateOnly

The second object to compare.

Returns

Boolean

`true` if left is the same as or earlier than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an instant in time, typically expressed as a date and time of day.

C#

```
public readonly struct DateTime : IComparable, IComparable<DateTime>,
IConvertible, IEquatable<DateTime>, IParsable<DateTime>, ISpanFormattable,
ISpanParsable<DateTime>, IUtf8SpanFormattable,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → DateTime

Implements IComparable , IComparable<DateTime> , IConvertible , IEquatable<DateTime> , IFormattable , ISerializable , ISpanFormattable , IParsable<DateTime> , IParsable<TSelf> , ISpanParsable<DateTime> , IUtf8SpanFormattable

Remarks

For more information about this API, see Supplemental API remarks for DateTime.

Constructors

Expand table

DateTime(DateOnly, TimeOnly)	Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly. The new instance will have the Unspecified kind.
DateTime(DateOnly, TimeOnly, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly and respecting the specified DateTimeKind.

DateTime(Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, and day.
DateTime(Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Date TimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32,	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond,

DateTimeKind)	and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int64)	Initializes a new instance of the DateTime structure to a specified number of ticks.
DateTime(Int64, DateTimeKind)	Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time.

Fields

Expand table

Max Value	Represents the largest possible value of DateTime. This field is read-only.
Min Value	Represents the smallest possible value of DateTime. This field is read-only.
Unix Epoch	The value of this constant is equivalent to 00:00:00.0000000 UTC, January 1, 1970, in the Gregorian calendar. UnixEpoch defines the point in time when Unix time is equal to 0.

Properties

Expand table

Date	Gets the date component of this instance.
Day	Gets the day of the month represented by this instance.
DayOfWeek	Gets the day of the week represented by this instance.
DayOfYear	Gets the day of the year represented by this instance.
Hour	Gets the hour component of the date represented by this instance.
Kind	Gets a value that indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither.
Microsecond	The microseconds component, expressed as a value between 0 and 999.
Millisecond	Gets the milliseconds component of the date represented by this instance.
Minute	Gets the minute component of the date represented by this instance.

Month	Gets the month component of the date represented by this instance.
Nanosecond	The nanoseconds component, expressed as a value between 0 and 900 (in increments of 100 nanoseconds).
Now	Gets a DateTime object that is set to the current date and time on this computer, expressed as the local time.
Second	Gets the seconds component of the date represented by this instance.
Ticks	Gets the number of ticks that represent the date and time of this instance.
TimeOfDay	Gets the time of day for this instance.
Today	Gets the current date.
UtcNow	Gets a DateTime object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC).
Year	Gets the year component of the date represented by this instance.

Methods

Expand table

Add(TimeSpan)	Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance.
AddDays(Double)	Returns a new DateTime that adds the specified number of days to the value of this instance.
AddHours(Double)	Returns a new DateTime that adds the specified number of hours to the value of this instance.
AddMicroseconds(Double)	Returns a new DateTime that adds the specified number of microseconds to the value of this instance.
AddMilliseconds(Double)	Returns a new DateTime that adds the specified number of milliseconds to the value of this instance.
AddMinutes(Double)	Returns a new DateTime that adds the specified number of minutes to the value of this instance.
AddMonths(Int32)	Returns a new DateTime that adds the specified number of months to the value of this instance.
AddSeconds(Double)	Returns a new DateTime that adds the specified number of seconds to the value of this instance.

AddTicks(Int64)	Returns a new DateTime that adds the specified number of ticks to the value of this instance.
AddYears(Int32)	Returns a new DateTime that adds the specified number of years to the value of this instance.
Compare(DateTime, DateTime)	Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance.
CompareTo(DateTime)	Compares the value of this instance to a specified DateTime value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.
CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified DateTime value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.
DaysInMonth(Int32, Int32)	Returns the number of days in the specified month and year.
Deconstruct(DateOnly, TimeOnly)	Deconstructs this DateTime instance by DateOnly and TimeOnly.
Deconstruct(Int32, Int32, Int32)	Deconstructs this DateOnly instance by Year, Month, and Day.
Equals(DateTime)	Returns a value indicating whether the value of this instance is equal to the value of the specified DateTime instance.
Equals(DateTime, DateTime)	Returns a value indicating whether two DateTime instances have the same date and time value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
FromBinary(Int64)	Deserializes a 64-bit binary value and recreates an original serialized DateTime object.
FromFileTime(Int64)	Converts the specified Windows file time to an equivalent local time.
FromFileTimeUtc(Int64)	Converts the specified Windows file time to an equivalent UTC time.
FromOADate(Double)	Returns a DateTime equivalent to the specified OLE Automation Date.

GetDateTimeFormats()	Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.
GetDateTimeFormats(Char)	Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier.
GetDateTimeFormats(Char, IFormatProvider)	Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier and culture-specific formatting information.
GetDateTimeFormats(IFormatProvider)	Converts the value of this instance to all the string representations supported by the standard date and time format specifiers and the specified culture-specific formatting information.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type DateTime.
IsDaylightSavingTime()	Indicates whether this instance of DateTime is within the daylight saving time range for the current time zone.
IsLeapYear(Int32)	Returns an indication whether the specified year is a leap year.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.
Parse(String)	Converts the string representation of a date and time to its DateTime equivalent by using the conventions of the current culture.
Parse(String, IFormatProvider)	Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format

	of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
SpecifyKind(DateTime, DateTimeKind)	Creates a new DateTime object that has the same number of ticks as the specified DateTime, but is designated as either local time, Coordinated Universal Time (UTC), or neither, as indicated by the specified DateTimeKind value.
Subtract(DateTime)	Returns a new TimeSpan that subtracts the specified date and time from the value of this instance.
Subtract(TimeSpan)	Returns a new DateTime that subtracts the specified duration from the value of this instance.
ToBinary()	Serializes the current DateTime object to a 64-bit binary value that subsequently can be used to recreate the DateTime object.
ToFileTime()	Converts the value of the current DateTime object to a Windows file time.
ToFileTimeUtc()	Converts the value of the current DateTime object to a Windows file time.
ToLocalTime()	Converts the value of the current DateTime object to local time.

ToLongDateString()	Converts the value of the current DateTime object to its equivalent long date string representation.
ToLongTimeString()	Converts the value of the current DateTime object to its equivalent long time string representation.
ToOADate()	Converts the value of this instance to the equivalent OLE Automation date.
ToShortDateString()	Converts the value of the current DateTime object to its equivalent short date string representation.
ToShortTimeString()	Converts the value of the current DateTime object to its equivalent short time string representation.
ToString()	Converts the value of the current DateTime object to its equivalent string representation using the formatting conventions of the current culture.
ToString(IFormatProvider)	Converts the value of the current DateTime object to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the value of the current DateTime object to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString(String, IFormatProvider)	Converts the value of the current DateTime object to its equivalent string representation using the specified format and culture-specific format information.
ToUniversalTime()	Converts the value of the current DateTime object to Coordinated Universal Time (UTC).
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current datetime instance into the provided span of characters.
TryParse(ReadOnlySpan<Char>, DateTime)	Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTime)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)	Converts the span representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.

TryParse(String, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateTime)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTime)	Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly. The method returns a value that indicates whether the conversion succeeded.

Operators

Expand table

Addition(DateTime, TimeSpan)	Adds a specified time interval to a specified date and time, yielding a new date and time.
Equality(DateTime, DateTime)	Determines whether two specified instances of DateTime are equal.

GreaterThan(DateTime, DateTime)	Determines whether one specified DateTime is later than another specified DateTime.
GreaterThanOrEqual(DateTime, DateTime)	Determines whether one specified DateTime represents a date and time that is the same as or later than another specified DateTime.
Inequality(DateTime, DateTime)	Determines whether two specified instances of DateTime are not equal.
LessThan(DateTime, DateTime)	Determines whether one specified DateTime is earlier than another specified DateTime.
LessThanOrEqual(DateTime, DateTime)	Determines whether one specified DateTime represents a date and time that is the same as or earlier than another specified DateTime.
Subtraction(DateTime, DateTime)	Subtracts a specified date and time from another specified date and time and returns a time interval.
Subtraction(DateTime, TimeSpan)	Subtracts a specified time interval from a specified date and time and returns a new date and time.

Explicit Interface Implementations

Expand table

IConvertible.ToBoolean(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToByte(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	Returns the current DateTime object.
IConvertible.ToDecimal(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDouble(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToInt16(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToInt32(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.

IConvertible.ToInt64(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToSByte(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToSingle(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToType(Type, IFormat Provider)	Converts the current DateTime object to an object of a specified type.
IConvertible.ToUInt16(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToUInt32(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToUInt64(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
ISerializable.GetObjectData(Serialization Info, StreamingContext)	Populates a SerializationInfo object with the data needed to serialize the current DateTime object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- DateTimeOffset

- TimeSpan
- Calendar
- GetUtcOffset(DateTime)
- TimeZoneInfo
- Choosing Between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo
- Working with Calendars
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the DateTime structure.

Overloads

Expand table

DateTime(Int64)	Initializes a new instance of the DateTime structure to a specified number of ticks.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time.

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second.
DateTime(Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar.
DateTime(Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, and day.
DateTime(DateOnly, TimeOnly, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly and respecting the specified DateTimeKind.
DateTime(Int64, DateTimeKind)	Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time.
DateTime(DateOnly, TimeOnly)	Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly. The new instance will have the Unspecified kind.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second for the specified calendar.

DateTime(Int64)

Initializes a new instance of the DateTime structure to a specified number of ticks.

C#

```
public DateTime (long ticks);
```

Parameters

`ticks` Int64

A date and time expressed in the number of 100-nanosecond intervals that have elapsed since January 1, 0001 at 00:00:00.000 in the Gregorian calendar.

Exceptions

ArgumentOutOfRangeException

`ticks` is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates one of the DateTime constructors.

C#

```
// This example demonstrates the DateTime(Int64) constructor.
using System;
using System.Globalization;

class Sample
{
    public static void Main()
    {
// Instead of using the implicit, default "G" date and time format
string, we
// use a custom format string that aligns the results and inserts leading
zeroes.
    string format = "{0}) The {1} date and time is {2:MM/dd/yyyy hh:mm:ss
tt}";

// Create a DateTime for the maximum date and time using ticks.
    DateTime dt1 = new DateTime(DateTime.MaxValue.Ticks);

// Create a DateTime for the minimum date and time using ticks.
    DateTime dt2 = new DateTime(DateTime.MinValue.Ticks);

// Create a custom DateTime for 7/28/1979 at 10:35:05 PM using a
// calendar based on the "en-US" culture, and ticks.
    long ticks = new DateTime(1979, 07, 28, 22, 35, 5,
    new CultureInfo("en-US", false).Calendar).Ticks;
    DateTime dt3 = new DateTime(ticks);

    Console.WriteLine(format, 1, "maximum", dt1);
    Console.WriteLine(format, 2, "minimum", dt2);
    Console.WriteLine(format, 3, "custom ", dt3);
    Console.WriteLine("\nThe custom date and time is created from {0:N0}
ticks.", ticks);
    }
}
/*
```

```
This example produces the following results:

1) The maximum date and time is 12/31/9999 11:59:59 PM
2) The minimum date and time is 01/01/0001 12:00:00 AM
3) The custom  date and time is 07/28/1979 10:35:05 PM

The custom date and time is created from 624,376,461,050,000,000 ticks.

*/
```

Remarks

The Kind property is initialized to Unspecified.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

```
C#
```

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, int microsecond, System.Globalization.Calendar
calendar);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

Exceptions

ArgumentNullException

`calendar` is `null`

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

Remarks

The allowable values for `year`, `month`, and `day` parameters depend on the `calendar` parameter. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

```
C#

public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, int microsecond, DateTimeKind kind);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

`kind` DateTimeKind

One of the enumeration values that indicates whether `year`, `month`, `day`, `hour`, `minute`, `second`, and `millisecond` specify a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

ArgumentException

`kind` is not one of the DateTimeKind values.

Remarks

This constructor interprets `year`, `month` and `day` as a year, month and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind) constructor.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

C#

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, System.Globalization.Calendar calendar,
DateTimeKind kind);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

`kind` DateTimeKind

One of the enumeration values that indicates whether `year`, `month`, `day`, `hour`, `minute`, `second`, and `millisecond` specify a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentNullException

`calendar` is `null`.

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

ArgumentException
`kind` is not one of the DateTimeKind values.

Examples

The following example calls the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind) constructor twice to instantiate two DateTime values. The first call instantiates a DateTime value by using a PersianCalendar object. Because the Persian calendar cannot be designated as the default calendar for a culture, displaying a date in the Persian calendar requires individual calls to its PersianCalendar.GetMonth, PersianCalendar.GetDayOfMonth, and PersianCalendar.GetYear methods. The second call to the constructor instantiates a DateTime value by using a HijriCalendar object. The example changes the current culture to Arabic (Syria) and changes the current culture's default calendar to the Hijri calendar. Because Hijri is the current culture's default calendar, the Console.WriteLine method uses it to format the date. When the previous current culture (which is English (United States) in this case) is restored, the Console.WriteLine method uses the current culture's default Gregorian calendar to format the date.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("Using the Persian Calendar:");
      PersianCalendar persian = new PersianCalendar();
      DateTime date1 = new DateTime(1389, 5, 27, 16, 32, 18, 500,
```

```
                                    persian, DateTimeKind.Local);
      Console.WriteLine("{0:M/dd/yyyy h:mm:ss.fff tt} {1}", date1,
date1.Kind);
      Console.WriteLine("{0}/{1}/{2} {3}{8}{4:D2}{8}{5:D2}.{6:G3} {7}\n",
                                        persian.GetMonth(date1),
                                        persian.GetDayOfMonth(date1),
                                        persian.GetYear(date1),
                                        persian.GetHour(date1),
                                        persian.GetMinute(date1),
                                        persian.GetSecond(date1),
                                        persian.GetMilliseconds(date1),
                                        date1.Kind,

DateTimeFormatInfo.CurrentInfo.TimeSeparator);

      Console.WriteLine("Using the Hijri Calendar:");
      // Get current culture so it can later be restored.
      CultureInfo dftCulture = Thread.CurrentThread.CurrentCulture;

      // Define strings for use in composite formatting.
      string dFormat;
      string fmtString;
      // Define Hijri calendar.
      HijriCalendar hijri = new HijriCalendar();
      // Make ar-SY the current culture and Hijri the current calendar.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("ar-SY");
      CultureInfo current = CultureInfo.CurrentCulture;
      current.DateTimeFormat.Calendar = hijri;
      dFormat = current.DateTimeFormat.ShortDatePattern;
      // Ensure year is displayed as four digits.
      dFormat = Regex.Replace(dFormat, "/yy$", "/yyyy") + " H:mm:ss.fff";
      fmtString = "{0} culture using the {1} calendar: {2:" + dFormat +
"} {3}";
      DateTime date2 = new DateTime(1431, 9, 9, 16, 32, 18, 500,
                                    hijri, DateTimeKind.Local);
      Console.WriteLine(fmtString, current, GetCalendarName(hijri),
                        date2, date2.Kind);

      // Restore previous culture.
      Thread.CurrentThread.CurrentCulture = dftCulture;
      dFormat = DateTimeFormatInfo.CurrentInfo.ShortDatePattern +"
H:mm:ss.fff";
      fmtString = "{0} culture using the {1} calendar: {2:" + dFormat +
"} {3}";
      Console.WriteLine(fmtString,
                        CultureInfo.CurrentCulture,

GetCalendarName(CultureInfo.CurrentCulture.Calendar),
                        date2, date2.Kind);
   }

   private static string GetCalendarName(Calendar cal)
   {
      return Regex.Match(cal.ToString(), "\\.
(\\w+)Calendar").Groups[1].Value;
```

```
    }
}
// The example displays the following output:
//    Using the Persian Calendar:
//    8/18/2010 4:32:18.500 PM Local
//    5/27/1389 16:32:18.500 Local
//
//    Using the Hijri Calendar:
//    ar-SY culture using the Hijri calendar: 09/09/1431 16:32:18.500
Local
//    en-US culture using the Gregorian calendar: 8/18/2010 16:32:18.500
Local
```

Remarks

The allowable values for `year`, `month`, and `day` parameters depend on the `calendar` parameter. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

The System.Globalization namespace provides several calendars including GregorianCalendar and JulianCalendar.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

C#

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, int microsecond);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

Remarks

This constructor interprets `year`, `month` and `day` as a year, month and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar) constructor.

The Kind property is initialized to Unspecified.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar.

C#

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, System.Globalization.Calendar calendar);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

Exceptions

ArgumentNullException

`calendar` is `null`.

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

Examples

The following example calls the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar) constructor twice to instantiate two DateTime values. The first call instantiates a DateTime value by using a PersianCalendar object. Because the Persian calendar cannot be designated as the default calendar for a culture, displaying a date in the Persian calendar requires individual calls to its PersianCalendar.GetMonth, PersianCalendar.GetDayOfMonth, and PersianCalendar.GetYear methods. The second call to the constructor instantiates a DateTime value by using a HijriCalendar object. The example changes the current culture to Arabic (Syria) and changes the current culture's default calendar to the Hijri calendar. Because Hijri is the current culture's default calendar, the Console.WriteLine method uses it to format the date. When the previous current culture (which is English (United States) in this case) is restored, the Console.WriteLine method uses the current culture's default Gregorian calendar to format the date.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("Using the Persian Calendar:");
      PersianCalendar persian = new PersianCalendar();
      DateTime date1 = new DateTime(1389, 5, 27, 16, 32, 18, 500,
persian);
      Console.WriteLine(date1.ToString("M/dd/yyyy h:mm:ss.fff tt"));
      Console.WriteLine("{0}/{1}/{2} {3}{7}{4:D2}{7}{5:D2}.{6:G3}\n",
                                       persian.GetMonth(date1),
                                       persian.GetDayOfMonth(date1),
                                       persian.GetYear(date1),
                                       persian.GetHour(date1),
                                       persian.GetMinute(date1),
                                       persian.GetSecond(date1),
                                       persian.GetMilliseconds(date1),

DateTimeFormatInfo.CurrentInfo.TimeSeparator);

      Console.WriteLine("Using the Hijri Calendar:");
      // Get current culture so it can later be restored.
      CultureInfo dftCulture = Thread.CurrentThread.CurrentCulture;

      // Define strings for use in composite formatting.
      string dFormat;
      string fmtString;
      // Define Hijri calendar.
      HijriCalendar hijri = new HijriCalendar();
```

```
      // Make ar-SY the current culture and Hijri the current calendar.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("ar-SY");
      CultureInfo current = CultureInfo.CurrentCulture;
      current.DateTimeFormat.Calendar = hijri;
      dFormat = current.DateTimeFormat.ShortDatePattern;
      // Ensure year is displayed as four digits.
      dFormat = Regex.Replace(dFormat, "/yy$", "/yyyy") + " H:mm:ss.fff";
      fmtString = "{0} culture using the {1} calendar: {2:" + dFormat +
"}";
      DateTime date2 = new DateTime(1431, 9, 9, 16, 32, 18, 500, hijri);
      Console.WriteLine(fmtString, current, GetCalendarName(hijri),
date2);

      // Restore previous culture.
      Thread.CurrentThread.CurrentCulture = dftCulture;
      dFormat = DateTimeFormatInfo.CurrentInfo.ShortDatePattern +"
H:mm:ss.fff";
      fmtString = "{0} culture using the {1} calendar: {2:" + dFormat +
"}";
      Console.WriteLine(fmtString,
                        CultureInfo.CurrentCulture,

GetCalendarName(CultureInfo.CurrentCulture.Calendar),
                        date2);
   }

   private static string GetCalendarName(Calendar cal)
   {
      return Regex.Match(cal.ToString(), "\\.
(\\w+)Calendar").Groups[1].Value;
   }
}
// The example displays the following output:
//       8/18/2010 4:32:18.500 PM
//       5/27/1389 16:32:18.500
//
//       Using the Hijri Calendar:
//       ar-SY culture using the Hijri calendar: 09/09/1431 16:32:18.500
//       en-US culture using the Gregorian calendar: 8/18/2010
16:32:18.500
```

Remarks

The Kind property is initialized to Unspecified.

The allowable values for `year`, `month`, and `day` depend on `calendar`. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

The System.Globalization namespace provides several calendars including GregorianCalendar and JulianCalendar.

See also

- Int32
- Calendar

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time.

```
C#

public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, DateTimeKind kind);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`kind` DateTimeKind

One of the enumeration values that indicates whether `year`, `month`, `day`, `hour`, `minute`, `second`, and `millisecond` specify a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

ArgumentException
`kind` is not one of the DateTimeKind values.

Examples

The following example uses the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind) constructor to instantiate a DateTime value.

C#

```
DateTime date1 = new DateTime(2010, 8, 18, 16, 32, 18, 500,
                              DateTimeKind.Local);
Console.WriteLine("{0:M/dd/yyyy h:mm:ss.fff tt} {1}", date1, date1.Kind);
// The example displays the following output, in this case for en-us
culture:
//      8/18/2010 4:32:18.500 PM Local
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month, and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind) constructor.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond.

C#

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

Examples

The following example uses the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32) constructor to instantiate a DateTime value.

C#

```
DateTime date1 = new DateTime(2010, 8, 18, 16, 32, 18, 500);
Console.WriteLine(date1.ToString("M/dd/yyyy h:mm:ss.fff tt"));
// The example displays the following output, in this case for en-us
culture:
//      8/18/2010 4:32:18.500 PM
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month, and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar) constructor.

The Kind property is initialized to Unspecified.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

```
C#

public DateTime (int year, int month, int day, int hour, int minute, int
second, int millisecond, int microsecond, System.Globalization.Calendar
calendar, DateTimeKind kind);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

`kind` DateTimeKind

One of the enumeration values that indicates whether `year`, `month`, `day`, `hour`, `minute`, `second`, and `millisecond` specify a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentNullException

`calendar` is `null`

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

ArgumentException

`kind` is not one of the DateTimeKind values.

Remarks

The allowable values for `year`, `month`, and `day` parameters depend on the `calendar` parameter. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time.

C#

```
public DateTime (int year, int month, int day, int hour, int minute, int
second, DateTimeKind kind);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`kind` DateTimeKind

One of the enumeration values that indicates whether `year`, `month`, `day`, `hour`, `minute` and `second` specify a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

ArgumentException

`kind` is not one of the DateTimeKind values.

Examples

The following example uses the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind) constructor to instantiate a DateTime value.

```
C#

DateTime date1 = new DateTime(2010, 8, 18, 16, 32, 0,
DateTimeKind.Local);
Console.WriteLine("{0} {1}", date1, date1.Kind);
// The example displays the following output, in this case for en-us
culture:
//      8/18/2010 4:32:00 PM Local
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month, and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind) constructor.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(Int32, Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second.

```
C#

public DateTime (int year, int month, int day, int hour, int minute, int
second);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

Examples

The following example uses the DateTime constructor to instantiate a DateTime value.

```
C#

DateTime date1 = new DateTime(2010, 8, 18, 16, 32, 0);
Console.WriteLine(date1.ToString());
// The example displays the following output, in this case for en-us
culture:
//      8/18/2010 4:32:00 PM
```

Remarks

The Kind property is initialized to Unspecified.

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month, and day in another calendar, call the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar) constructor.

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(Int32, Int32, Int32, Calendar)

Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar.

C#

```
public DateTime (int year, int month, int day,
System.Globalization.Calendar calendar);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

Exceptions

ArgumentNullException

`calendar` is `null`.

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

Examples

The following example calls the DateTime(Int32, Int32, Int32, Calendar) constructor twice to instantiate two DateTime values. The first call instantiates a DateTime value by using a PersianCalendar object. Because the Persian calendar cannot be designated as the default calendar for a culture, displaying a date in the Persian calendar requires individual calls to its PersianCalendar.GetMonth, PersianCalendar.GetDayOfMonth, and PersianCalendar.GetYear methods. The second call to the constructor instantiates a DateTime value by using a HijriCalendar object. The example changes the current culture to Arabic (Syria) and changes the current culture's default calendar to the Hijri calendar. Because Hijri is the current culture's default calendar, the Console.WriteLine method uses it to format the date. When the previous current culture (which is English (United States) in this case) is restored, the Console.WriteLine method uses the current culture's default Gregorian calendar to format the date.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("Using the Persian Calendar:");
      PersianCalendar persian = new PersianCalendar();
      DateTime date1 = new DateTime(1389, 5, 27, persian);
      Console.WriteLine(date1.ToString());
      Console.WriteLine("{0}/{1}/{2}\n", persian.GetMonth(date1),
                                         persian.GetDayOfMonth(date1),
                                         persian.GetYear(date1));

      Console.WriteLine("Using the Hijri Calendar:");
      // Get current culture so it can later be restored.
      CultureInfo dftCulture = Thread.CurrentThread.CurrentCulture;

      // Define Hijri calendar.
```

```
        HijriCalendar hijri = new HijriCalendar();
        // Make ar-SY the current culture and Hijri the current calendar.
        Thread.CurrentThread.CurrentCulture = new CultureInfo("ar-SY");
        CultureInfo current = CultureInfo.CurrentCulture;
        current.DateTimeFormat.Calendar = hijri;
        string dFormat = current.DateTimeFormat.ShortDatePattern;
        // Ensure year is displayed as four digits.
        dFormat = Regex.Replace(dFormat, "/yy$", "/yyyy");
        current.DateTimeFormat.ShortDatePattern = dFormat;
        DateTime date2 = new DateTime(1431, 9, 9, hijri);
        Console.WriteLine("{0} culture using the {1} calendar: {2:d}",
current,
                          GetCalendarName(hijri), date2);

        // Restore previous culture.
        Thread.CurrentThread.CurrentCulture = dftCulture;
        Console.WriteLine("{0} culture using the {1} calendar: {2:d}",
                          CultureInfo.CurrentCulture,

GetCalendarName(CultureInfo.CurrentCulture.Calendar),
                          date2);
    }

    private static string GetCalendarName(Calendar cal)
    {
        return Regex.Match(cal.ToString(), "\\.
(\\w+)Calendar").Groups[1].Value;
    }
}
// The example displays the following output:
//       Using the Persian Calendar:
//       8/18/2010 12:00:00 AM
//       5/27/1389
//
//       Using the Hijri Calendar:
//       ar-SY culture using the Hijri calendar: 09/09/1431
//       en-US culture using the Gregorian calendar: 8/18/2010
```

Remarks

The time of day for the resulting DateTime is midnight (00:00:00). The Kind property is initialized to Unspecified.

The allowable values for `year`, `month`, and `day` depend on `calendar`. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning

of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

The System.Globalization namespace provides several calendars including GregorianCalendar and JulianCalendar.

See also

- Int32
- Calendar

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DateTime(Int32, Int32, Int32)

Initializes a new instance of the DateTime structure to the specified year, month, and day.

C#

```
public DateTime (int year, int month, int day);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

Examples

The following example uses the DateTime(Int32, Int32, Int32) constructor to instantiate a DateTime value. The example also illustrates that this overload creates a DateTime value whose time component equals midnight (or 0:00).

C#

```
DateTime date1 = new DateTime(2010, 8, 18);
Console.WriteLine(date1.ToString());
// The example displays the following output:
//       8/18/2010 12:00:00 AM
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTime value by using the year, month, and day in another calendar, call the DateTime(Int32, Int32, Int32, Calendar) constructor.

The time of day for the resulting DateTime is midnight (00:00:00). The Kind property is initialized to DateTimeKind.Unspecified.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(DateOnly, TimeOnly, DateTimeKind)

Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly and respecting the specified DateTimeKind.

C#

```
public DateTime (DateOnly date, TimeOnly time, DateTimeKind kind);
```

Parameters

`date` DateOnly

The date part.

`time` TimeOnly

The time part.

`kind` DateTimeKind

One of the enumeration values that indicates whether `date` and `time` specify a local time, Coordinated Universal Time (UTC), or neither.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

DateTime(Int64, DateTimeKind)

Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time.

```
C#
public DateTime (long ticks, DateTimeKind kind);
```

Parameters

`ticks` Int64

A date and time expressed in the number of 100-nanosecond intervals that have elapsed since January 1, 0001 at 00:00:00.000 in the Gregorian calendar.

`kind` DateTimeKind

One of the enumeration values that indicates whether `ticks` specifies a local time, Coordinated Universal Time (UTC), or neither.

Exceptions

ArgumentOutOfRangeException

`ticks` is less than DateTime.MinValue or greater than DateTime.MaxValue.

ArgumentException

`kind` is not one of the DateTimeKind values.

Remarks

For applications in which portability of date and time data or a limited degree of time zone awareness is important, you can use the corresponding DateTimeOffset constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTime(DateOnly, TimeOnly)

Initializes a new instance of the DateTime structure to the specified DateOnly and TimeOnly. The new instance will have the Unspecified kind.

C#

```
public DateTime (DateOnly date, TimeOnly time);
```

Parameters

`date` DateOnly

The date part.

`time` TimeOnly

The time part.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar)

Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second for the specified calendar.

```
C#

public DateTime (int year, int month, int day, int hour, int minute, int
second, System.Globalization.Calendar calendar);
```

Parameters

`year` Int32

The year (1 through the number of years in `calendar`).

`month` Int32

The month (1 through the number of months in `calendar`).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

Exceptions

ArgumentNullException

`calendar` is `null`.

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

Examples

The following example calls the DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar) constructor twice to instantiate two DateTime values. The first call instantiates a DateTime value by using a PersianCalendar object. Because the Persian calendar cannot be designated as the default calendar for a culture, displaying a date in the Persian calendar requires individual calls to its PersianCalendar.GetMonth, PersianCalendar.GetDayOfMonth, and PersianCalendar.GetYear methods. The second call to the constructor instantiates a DateTime value by using a HijriCalendar object. The example changes the current culture to Arabic (Syria) and changes the current culture's default calendar to the Hijri calendar. Because Hijri is the current culture's default calendar, the Console.WriteLine method uses it to format the date. When the previous current culture (which is English (United States) in this case) is restored, the Console.WriteLine method uses the current culture's default Gregorian calendar to format the date.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("Using the Persian Calendar:");
```

```
      PersianCalendar persian = new PersianCalendar();
      DateTime date1 = new DateTime(1389, 5, 27, 16, 32, 0, persian);
      Console.WriteLine(date1.ToString());
      Console.WriteLine("{0}/{1}/{2} {3}{6}{4:D2}{6}{5:D2}\n",
                                        persian.GetMonth(date1),
                                        persian.GetDayOfMonth(date1),
                                        persian.GetYear(date1),
                                        persian.GetHour(date1),
                                        persian.GetMinute(date1),
                                        persian.GetSecond(date1),

DateTimeFormatInfo.CurrentInfo.TimeSeparator);

      Console.WriteLine("Using the Hijri Calendar:");
      // Get current culture so it can later be restored.
      CultureInfo dftCulture = Thread.CurrentThread.CurrentCulture;

      // Define Hijri calendar.
      HijriCalendar hijri = new HijriCalendar();
      // Make ar-SY the current culture and Hijri the current calendar.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("ar-SY");
      CultureInfo current = CultureInfo.CurrentCulture;
      current.DateTimeFormat.Calendar = hijri;
      string dFormat = current.DateTimeFormat.ShortDatePattern;
      // Ensure year is displayed as four digits.
      dFormat = Regex.Replace(dFormat, "/yy$", "/yyyy");
      current.DateTimeFormat.ShortDatePattern = dFormat;
      DateTime date2 = new DateTime(1431, 9, 9, 16, 32, 18, hijri);
      Console.WriteLine("{0} culture using the {1} calendar: {2:g}",
current,
                        GetCalendarName(hijri), date2);

      // Restore previous culture.
      Thread.CurrentThread.CurrentCulture = dftCulture;
      Console.WriteLine("{0} culture using the {1} calendar: {2:g}",
                        CultureInfo.CurrentCulture,

GetCalendarName(CultureInfo.CurrentCulture.Calendar),
                        date2);
   }

   private static string GetCalendarName(Calendar cal)
   {
      return Regex.Match(cal.ToString(), "\\.
(\\w+)Calendar").Groups[1].Value;
   }
}
// The example displays the following output:
//       Using the Persian Calendar:
//       8/18/2010 4:32:00 PM
//       5/27/1389 16:32:00
//
//       Using the Hijri Calendar:
//       ar-SY culture using the Hijri calendar: 09/09/1431 04:32 م
//       en-US culture using the Gregorian calendar: 8/18/2010 4:32 PM
```

Remarks

The Kind property is initialized to Unspecified.

The allowable values for `year`, `month`, and `day` depend on `calendar`. An exception is thrown if the specified date and time cannot be expressed using `calendar`.

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

The System.Globalization namespace provides several calendars including GregorianCalendar and JulianCalendar.

See also

- Int32
- Calendar

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of DateTime. This field is read-only.

C#

```
public static readonly DateTime MaxValue;
```

Field Value

DateTime

Examples

The following example instantiates a DateTime object by passing its constructor an Int64 value that represents a number of ticks. Before invoking the constructor, the example ensures that this value is greater than or equal to `DateTime.MinValue.Ticks` and less than or equal to `DateTime.MaxValue.Ticks`. If not, it throws an ArgumentOutOfRangeException.

C#

```
// Attempt to assign an out-of-range value to a DateTime constructor.
long numberOfTicks = Int64.MaxValue;
DateTime validDate;

// Validate the value.
if (numberOfTicks >= DateTime.MinValue.Ticks &&
    numberOfTicks <= DateTime.MaxValue.Ticks)
   validDate = new DateTime(numberOfTicks);
else if (numberOfTicks < DateTime.MinValue.Ticks)
   Console.WriteLine("{0:N0} is less than {1:N0} ticks.",
                     numberOfTicks,
                     DateTime.MinValue.Ticks);
else
   Console.WriteLine("{0:N0} is greater than {1:N0} ticks.",
                     numberOfTicks,
```

```
                        DateTime.MaxValue.Ticks);
// The example displays the following output:
//   9,223,372,036,854,775,807 is greater than 3,155,378,975,999,999,999
ticks.
```

Remarks

The value of this constant is equivalent to 23:59:59.9999999 UTC, December 31, 9999 in the Gregorian calendar, exactly one 100-nanosecond tick before 00:00:00 UTC, January 1, 10000.

Some calendars, such as the ThaiBuddhistCalendar, support an upper date range that is earlier than MaxValue. In these cases, trying to access MaxValue in variable assignments or formatting and parsing operations can throw an ArgumentOutOfRangeException. Rather than retrieving the value of DateTime.MaxValue, you can retrieve the value of the specified culture's latest valid date value from the Calendar.MaxSupportedDateTime property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTime.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of DateTime. This field is read-only.

C#

```
public static readonly DateTime MinValue;
```

Field Value

DateTime

Examples

The following example instantiates a DateTime object by passing its constructor an Int64 value that represents a number of ticks. Before invoking the constructor, the example ensures that this value is greater than or equal to `DateTime.MinValue.Ticks` and less than or equal to `DateTime.MaxValue.Ticks`. If not, it throws an ArgumentOutOfRangeException.

C#

```
// Attempt to assign an out-of-range value to a DateTime constructor.
long numberOfTicks = Int64.MaxValue;
DateTime validDate;

// Validate the value.
if (numberOfTicks >= DateTime.MinValue.Ticks &&
    numberOfTicks <= DateTime.MaxValue.Ticks)
   validDate = new DateTime(numberOfTicks);
else if (numberOfTicks < DateTime.MinValue.Ticks)
   Console.WriteLine("{0:N0} is less than {1:N0} ticks.",
                     numberOfTicks,
                     DateTime.MinValue.Ticks);
else
   Console.WriteLine("{0:N0} is greater than {1:N0} ticks.",
                     numberOfTicks,
```

```
                        DateTime.MaxValue.Ticks);
// The example displays the following output:
//   9,223,372,036,854,775,807 is greater than 3,155,378,975,999,999,999
ticks.
```

Remarks

The value of this constant is equivalent to 00:00:00.0000000 UTC, January 1, 0001, in the Gregorian calendar.

MinValue defines the date and time that is assigned to an uninitialized DateTime variable. The following example illustrates this.

C#

```
// Define an uninitialized date.
DateTime date1 = new DateTime();
Console.Write(date1);
if (date1.Equals(DateTime.MinValue))
   Console.WriteLine("  (Equals Date.MinValue)");
// The example displays the following output:
//    1/1/0001 12:00:00 AM  (Equals Date.MinValue)
```

The MinValue and MaxValue properties can be used to ensure that a value lies within the supported range before passing it to a DateTime constructor. The code in the Example section illustrates this usage.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTime.UnixEpoch Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The value of this constant is equivalent to 00:00:00.0000000 UTC, January 1, 1970, in the Gregorian calendar. UnixEpoch defines the point in time when Unix time is equal to 0.

C#

```
public static readonly DateTime UnixEpoch;
```

Field Value

DateTime

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Date Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the date component of this instance.

C#

```
public DateTime Date { get; }
```

Property Value

DateTime

A new object with the same date as this instance, and the time value set to 12:00:00 midnight (00:00:00).

Examples

The following example uses the Date property to extract the date component of a DateTime value with its time component set to zero (or 0:00:00, or midnight). It also illustrates that, depending on the format string used when displaying the DateTime value, the time component can continue to appear in formatted output.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime date1 = new DateTime(2008, 6, 1, 7, 47, 0);
      Console.WriteLine(date1.ToString());

      // Get date-only portion of date, without its time.
      DateTime dateOnly = date1.Date;
      // Display date using short date string.
      Console.WriteLine(dateOnly.ToString("d"));
```

```
        // Display date using 24-hour clock.
        Console.WriteLine(dateOnly.ToString("g"));
        Console.WriteLine(dateOnly.ToString("MM/dd/yyyy HH:mm"));
    }
}
// The example displays output like the following output:
//       6/1/2008 7:47:00 AM
//       6/1/2008
//       6/1/2008 12:00 AM
//       06/01/2008 00:00
```

Remarks

The value of the Kind property of the returned DateTime value is the same as that of the current instance.

Because the DateTime type represents both dates and times in a single type, it is important to avoid misinterpreting a date returned by the Date property as a date and time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Day Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the month represented by this instance.

C#

```
public int Day { get; }
```

Property Value

Int32

The day component, expressed as a value between 1 and 31.

Examples

The following example demonstrates the Day property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Remarks

The Day property always returns the day of the month in the Gregorian calendar, even if the current DateTime value was instantiated using some other calendar or if the current culture's default calendar is not the Gregorian calendar. To retrieve the day of the month of a particular date using some other calendar, call that calendar's Calendar.GetDayOfMonth method. The following example uses both the Day property and the HijriCalendar.GetDayOfMonth method to retrieve the day of the month for a DateTime value that is instantiated using the Hijri calendar.

C#

```
// Return day of 1/13/2009.
DateTime dateGregorian = new DateTime(2009, 1, 13);
Console.WriteLine(dateGregorian.Day);
// Displays 13 (Gregorian day).

// Create date of 1/13/2009 using Hijri calendar.
HijriCalendar hijri = new HijriCalendar();
DateTime dateHijri = new DateTime(1430, 1, 17, hijri);
// Return day of date created using Hijri calendar.
Console.WriteLine(dateHijri.Day);
// Displays 13 (Gregorian day).

// Display day of date in Hijri calendar.
Console.WriteLine(hijri.GetDayOfMonth(dateHijri));
// Displays 17 (Hijri day).
```

Similarly, the following example uses both the Day property and the HijriCalendar.GetDayOfMonth method to retrieve the day of the month when the current culture is ar-SA, which uses Hijri as its default calendar.

C#

```
CultureInfo originalCulture = Thread.CurrentThread.CurrentCulture;

// Change current culture to ar-SA.
CultureInfo ci = new CultureInfo("ar-SA");
Thread.CurrentThread.CurrentCulture = ci;

DateTime hijriDate = new DateTime(1430, 1, 17,
                          Thread.CurrentThread.CurrentCulture.Calendar);
// Display date (uses calendar of current culture by default).
```

```
Console.WriteLine(hijriDate.ToString("dd-MM-yyyy"));
// Displays 17-01-1430.

// Display day of 17th of Muharram
Console.WriteLine(hijriDate.Day);
// Displays 13 (corresponding day of January in Gregorian calendar).

// Display day of 17th of Muharram in Hijri calendar.
Console.WriteLine(Thread.CurrentThread.CurrentCulture.Calendar.GetDayOfMonth
(hijriDate));
// Displays 17.

Thread.CurrentThread.CurrentCulture = originalCulture;
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.DayOfWeek Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the week represented by this instance.

C#

```
public DayOfWeek DayOfWeek { get; }
```

Property Value

DayOfWeek

An enumerated constant that indicates the day of the week of this DateTime value.

Examples

The following example demonstrates the DayOfWeek property and the System.DayOfWeek enumeration.

C#

```
// This example demonstrates the DateTime.DayOfWeek property
using System;

class Sample
{
    public static void Main()
    {
// Assume the current culture is en-US.
// Create a DateTime for the first of May, 2003.
    DateTime dt = new DateTime(2003, 5, 1);
    Console.WriteLine("Is Thursday the day of the week for {0:d}?: {1}",
                      dt, dt.DayOfWeek == DayOfWeek.Thursday);
    Console.WriteLine("The day of the week for {0:d} is {1}.", dt,
dt.DayOfWeek);
    }
}
/*
```

```
This example produces the following results:

Is Thursday the day of the week for 5/1/2003?: True
The day of the week for 5/1/2003 is Thursday.
*/
```

Remarks

The value of the constants in the DayOfWeek enumeration ranges from DayOfWeek.Sunday to DayOfWeek.Saturday. If cast to an integer, its value ranges from zero (which indicates DayOfWeek.Sunday) to six (which indicates DayOfWeek.Saturday).

The DayOfWeek property returns an enumerated constant; it does not reflect a system's regional and language settings. To retrieve a string representing a localized weekday name for a particular date, call one of the overloads of the ToString method that includes a `format` parameter and pass it either the `ddd` or `dddd` custom format strings. For details, see How to: Extract the Day of the Week from a Specific Date.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Ticks
- Int32

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTime.DayOfYear Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the year represented by this instance.

C#

```
public int DayOfYear { get; }
```

Property Value

Int32

The day of the year, expressed as a value between 1 and 366.

Examples

The following example displays the day of the year of December 31 for the years 2010-2020 in the Gregorian calendar. Note that the example shows that December 31 is the 366th day of the year in leap years.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime dec31 = new DateTime(2010, 12, 31);
      for (int ctr = 0; ctr <= 10; ctr++) {
         DateTime dateToDisplay = dec31.AddYears(ctr);
         Console.WriteLine("{0:d}: day {1} of {2} {3}", dateToDisplay,
                           dateToDisplay.DayOfYear,
                           dateToDisplay.Year,
                           DateTime.IsLeapYear(dateToDisplay.Year) ?
                                              "(Leap Year)" : "");
      }
   }
```

```
}
// The example displays the following output:
//       12/31/2010: day 365 of 2010
//       12/31/2011: day 365 of 2011
//       12/31/2012: day 366 of 2012 (Leap Year)
//       12/31/2013: day 365 of 2013
//       12/31/2014: day 365 of 2014
//       12/31/2015: day 365 of 2015
//       12/31/2016: day 366 of 2016 (Leap Year)
//       12/31/2017: day 365 of 2017
//       12/31/2018: day 365 of 2018
//       12/31/2019: day 365 of 2019
//       12/31/2020: day 366 of 2020 (Leap Year)
```

Remarks

The DayOfYear property takes leap years into account when it calculates the day of the year. The property value always reflects the day of the year in the Gregorian calendar, regardless of the current culture's current calendar. To retrieve the day of the year in a different calendar, call the Calendar.GetDayOfYear method of that calendar.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetDayOfYear(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTime.Hour Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hour component of the date represented by this instance.

C#

```
public int Hour { get; }
```

Property Value

Int32

The hour component, expressed as a value between 0 and 23.

Examples

The following example demonstrates the Hour property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Remarks

The value of the Hour property is always expressed using a 24-hour clock. To retrieve a string that represents the hour of a date and time using a 12-hour clock, call the DateTime.ToString(String) or DateTime.ToString(String, IFormatProvider) method with the "h" custom format specifier. For example:

C#

```
DateTime date1 = new DateTime(2008, 4, 1, 18, 53, 0);
Console.WriteLine(date1.ToString("%h"));              // Displays 6
Console.WriteLine(date1.ToString("h tt"));            // Displays 6 PM
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Kind Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither.

C#

```
public DateTimeKind Kind { get; }
```

Property Value

DateTimeKind

One of the enumeration values that indicates what the current time represents. The default is Unspecified.

Examples

The following example uses the SpecifyKind method to demonstrate how the Kind property influences the ToLocalTime and ToUniversalTime conversion methods.

C#

```
// This code example demonstrates the DateTime Kind, Now, and
// UtcNow properties, and the SpecifyKind(), ToLocalTime(),
// and ToUniversalTime() methods.

using System;

class Sample
{
    public static void Main()
    {
        // Get the date and time for the current moment, adjusted
        // to the local time zone.

        DateTime saveNow = DateTime.Now;
```

```
        // Get the date and time for the current moment expressed
        // as coordinated universal time (UTC).

        DateTime saveUtcNow = DateTime.UtcNow;
        DateTime myDt;

        // Display the value and Kind property of the current moment
        // expressed as UTC and local time.

        DisplayNow("UtcNow: ..........", saveUtcNow);
        DisplayNow("Now: ............", saveNow);
        Console.WriteLine();

        // Change the Kind property of the current moment to
        // DateTimeKind.Utc and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Utc);
        Display("Utc: ............", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Local and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Local);
        Display("Local: ...........", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Unspecified and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Unspecified);
        Display("Unspecified: .....", myDt);
    }

    // Display the value and Kind property of a DateTime structure, the
    // DateTime structure converted to local time, and the DateTime
    // structure converted to universal time.

    public static string datePatt = @"M/d/yyyy hh:mm:ss tt";
    public static void Display(string title, DateTime inputDt)
    {
        DateTime dispDt = inputDt;
        string dtString;

        // Display the original DateTime.

        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, dispDt.Kind);

        // Convert inputDt to local time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was universal time.
```

```
        dispDt = inputDt.ToLocalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToLocalTime:     {0}, Kind = {1}",
                          dtString, dispDt.Kind);

        // Convert inputDt to universal time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was local time.

        dispDt = inputDt.ToUniversalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToUniversalTime: {0}, Kind = {1}",
                          dtString, dispDt.Kind);
        Console.WriteLine();
    }

    // Display the value and Kind property for DateTime.Now and
DateTime.UtcNow.

    public static void DisplayNow(string title, DateTime inputDt)
    {
        string dtString = inputDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, inputDt.Kind);
    }
}

/*
This code example produces the following results:

UtcNow: ........... 5/6/2005 09:34:42 PM, Kind = Utc
Now: ............. 5/6/2005 02:34:42 PM, Kind = Local

Utc: ............. 5/6/2005 02:34:42 PM, Kind = Utc
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 02:34:42 PM, Kind = Utc

Local: ........... 5/6/2005 02:34:42 PM, Kind = Local
  ToLocalTime:     5/6/2005 02:34:42 PM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

Unspecified: ..... 5/6/2005 02:34:42 PM, Kind = Unspecified
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

*/
```

Remarks

You can explicitly set the Kind property of a new DateTime value to a particular DateTimeKind value by calling the SpecifyKind method.

The Kind property allows a DateTime value to clearly reflect either Coordinated Universal Time (UTC) or the local time. In contrast, the DateTimeOffset structure can unambiguously reflect any time in any time zone as a single point in time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- SpecifyKind(DateTime, DateTimeKind)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Microsecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The microseconds component, expressed as a value between 0 and 999.

C#

```
public int Microsecond { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Millisecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the milliseconds component of the date represented by this instance.

C#

```
public int Millisecond { get; }
```

Property Value

Int32

The milliseconds component, expressed as a value between 0 and 999.

Examples

The following example demonstrates the Millisecond property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Remarks

You can display the string representation of the Millisecond property by using the "fff" format specifier. For example, the following code displays a string that contains the number of milliseconds in a date and time to the console.

C#

```
DateTime date1 = new DateTime(2008, 1, 1, 0, 30, 45, 125);
Console.WriteLine("Milliseconds: {0:fff}",
                  date1);           // displays Milliseconds: 125
```

You can also display the millisecond component together with the other components of a date and time value by using the "o" standard format specifier. For example:

C#

```
DateTime date2 = new DateTime(2008, 1, 1, 0, 30, 45, 125);
Console.WriteLine("Date: {0:o}",
                  date2);
// Displays the following output to the console:
//      Date: 2008-01-01T00:30:45.1250000
```

However, the "o" format specifier is intended less for displaying than for round-tripping or storing a DateTime value. You can also display milliseconds together with other date and time components by using a custom format string, as the following example shows.

C#

```
DateTime date3 = new DateTime(2008, 1, 1, 0, 30, 45, 125);
Console.WriteLine("Date with milliseconds: {0:MM/dd/yyy HH:mm:ss.fff}",
                  date3);
// Displays the following output to the console:
//       Date with milliseconds: 01/01/2008 00:30:45.125
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Minute Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minute component of the date represented by this instance.

C#

```
public int Minute { get; }
```

Property Value

Int32

The minute component, expressed as a value between 0 and 59.

Examples

The following example demonstrates the Minute property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Month Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the month component of the date represented by this instance.

C#

```
public int Month { get; }
```

Property Value

Int32

The month component, expressed as a value between 1 and 12.

Examples

The following example demonstrates the Month property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Nanosecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The nanoseconds component, expressed as a value between 0 and 900 (in increments of 100 nanoseconds).

C#

```
public int Nanosecond { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Now Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime object that is set to the current date and time on this computer, expressed as the local time.

C#

```
public static DateTime Now { get; }
```

Property Value

DateTime

An object whose value is the current local date and time.

Examples

The following example uses the Now and UtcNow properties to retrieve the current local date and time and the current universal coordinated (UTC) date and time. It then uses the formatting conventions of a number of cultures to display the strings, along with the values of their Kind properties.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      DateTime localDate = DateTime.Now;
      DateTime utcDate = DateTime.UtcNow;
      String[] cultureNames = { "en-US", "en-GB", "fr-FR",
                                "de-DE", "ru-RU" } ;

      foreach (var cultureName in cultureNames) {
```

```
            var culture = new CultureInfo(cultureName);
            Console.WriteLine("{0}:", culture.NativeName);
            Console.WriteLine("   Local date and time: {0}, {1:G}",
                              localDate.ToString(culture), localDate.Kind);
            Console.WriteLine("   UTC date and time: {0}, {1:G}\n",
                              utcDate.ToString(culture), utcDate.Kind);
        }
    }
}
// The example displays the following output:
//       English (United States):
//          Local date and time: 6/19/2015 10:35:50 AM, Local
//          UTC date and time: 6/19/2015 5:35:50 PM, Utc
//
//       English (United Kingdom):
//          Local date and time: 19/06/2015 10:35:50, Local
//          UTC date and time: 19/06/2015 17:35:50, Utc
//
//       français (France):
//          Local date and time: 19/06/2015 10:35:50, Local
//          UTC date and time: 19/06/2015 17:35:50, Utc
//
//       Deutsch (Deutschland):
//          Local date and time: 19.06.2015 10:35:50, Local
//          UTC date and time: 19.06.2015 17:35:50, Utc
//
//       русский (Россия):
//          Local date and time: 19.06.2015 10:35:50, Local
//          UTC date and time: 19.06.2015 17:35:50, Utc
```

Remarks

The Now property returns a DateTime value that represents the current date and time on the local computer. Note that there is a difference between a DateTime value, which represents the number of ticks that have elapsed since midnight of January 1, 0001, and the string representation of that DateTime value, which expresses a date and time value in a culture-specific-specific format. For information on formatting date and time values, see the ToString method. The following example displays the short date and time string in a number of culture-specific formats.

C#

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
```

```
        DateTime localDate = DateTime.Now;
        String[] cultureNames = { "en-US", "en-GB", "fr-FR",
                                  "de-DE", "ru-RU" };

        foreach (var cultureName in cultureNames) {
           var culture = new CultureInfo(cultureName);
           Console.WriteLine("{0}: {1}", cultureName,
                             localDate.ToString(culture));
        }
    }
}
// The example displays the following output:
//       en-US: 6/19/2015 10:03:06 AM
//       en-GB: 19/06/2015 10:03:06
//       fr-FR: 19/06/2015 10:03:06
//       de-DE: 19.06.2015 10:03:06
//       ru-RU: 19.06.2015 10:03:06
```

The resolution of this property depends on the system timer, which depends on the underlying operating system. It tends to be between 0.5 and 15 milliseconds. As a result, repeated calls to the Now property in a short time interval, such as in a loop, may return the same value.

The Now property is frequently used to measure performance. However, because of its low resolution, it is not suitable for use as a benchmarking tool. A better alternative is to use the Stopwatch class.

Starting with the .NET Framework version 2.0, the return value is a DateTime whose Kind property returns DateTimeKind.Local.

> ⓘ **Note**
>
> You can also use the **DateTimeOffset.Now** property to retrieve the current local date and time. It allows a local time to be expressed unambiguously as a single point in time, which in turn makes that time value portable across computers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Now
- UtcNow

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Second Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the seconds component of the date represented by this instance.

C#

```
public int Second { get; }
```

Property Value

Int32

The seconds component, expressed as a value between 0 and 59.

Examples

The following example demonstrates the Second property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Ticks Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of ticks that represent the date and time of this instance.

C#

```
public long Ticks { get; }
```

Property Value

Int64

The number of ticks that represent the date and time of this instance. The value is between `DateTime.MinValue.Ticks` and `DateTime.MaxValue.Ticks`.

Examples

The following example uses the Ticks property to display the number of ticks that have elapsed since the beginning of the twenty-first century and to instantiate a TimeSpan object. The TimeSpan object is then used to display the elapsed time using several other time intervals.

C#

```
DateTime centuryBegin = new DateTime(2001, 1, 1);
DateTime currentDate = DateTime.Now;

long elapsedTicks = currentDate.Ticks - centuryBegin.Ticks;
TimeSpan elapsedSpan = new TimeSpan(elapsedTicks);

Console.WriteLine("Elapsed from the beginning of the century to {0:f}:",
                  currentDate);
Console.WriteLine("   {0:N0} nanoseconds", elapsedTicks * 100);
Console.WriteLine("   {0:N0} ticks", elapsedTicks);
Console.WriteLine("   {0:N2} seconds", elapsedSpan.TotalSeconds);
Console.WriteLine("   {0:N2} minutes", elapsedSpan.TotalMinutes);
Console.WriteLine("   {0:N0} days, {1} hours, {2} minutes, {3} seconds",
```

```
                    elapsedSpan.Days, elapsedSpan.Hours,
                    elapsedSpan.Minutes, elapsedSpan.Seconds);

// This example displays an output similar to the following:
//
// Elapsed from the beginning of the century to Thursday, 14 November 2019
18:21:
//    595,448,498,171,000,000 nanoseconds
//    5,954,484,981,710,000 ticks
//    595,448,498.17 seconds
//    9,924,141.64 minutes
//    6,891 days, 18 hours, 21 minutes, 38 seconds
```

Remarks

A single tick represents one hundred nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond (see TicksPerMillisecond) and 10 million ticks in a second.

The value of this property represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight, January 1, 0001 in the Gregorian calendar, which represents MinValue. It does not include the number of ticks that are attributable to leap seconds. If the DateTime object has its `Kind` property set to `Local`, its ticks represent the time elapsed time since 12:00:00 midnight, January 1, 0001 in the local time as specified by the current time zone setting. If the `DateTime` object has its `Kind` property set to `Utc`, its ticks represent the time elapsed time since 12:00:00 midnight, January 1, 0001 in the Coordinated Universal Time. If the `DateTime` object has its `Kind` property set to `Unspecified`, its ticks represent the time elapsed time since 12:00:00 midnight, January 1, 0001 in the unknown time zone.

In general, the ticks represent the time according to the time zone specified by the `Kind` property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.TimeOfDay Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time of day for this instance.

C#

```
public TimeSpan TimeOfDay { get; }
```

Property Value

TimeSpan

A time interval that represents the fraction of the day that has elapsed since midnight.

Examples

The following example displays the value of the TimeOfDay property for an array of DateTime values. It also contrasts the return value with the string returned by the "t" standard format string in a composite formatting operation.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime[] dates = { DateTime.Now,
                           new DateTime(2013, 9, 14, 9, 28, 0),
                           new DateTime(2011, 5, 28, 10, 35, 0),
                           new DateTime(1979, 12, 25, 14, 30, 0) };
      foreach (var date in dates) {
         Console.WriteLine("Day: {0:d} Time: {1:g}", date.Date,
date.TimeOfDay);
         Console.WriteLine("Day: {0:d} Time: {0:t}\n", date);
      }
   }
```

```
}
// The example displays output like the following:
//    Day: 7/25/2012 Time: 10:08:12.9713744
//    Day: 7/25/2012 Time: 10:08 AM
//
//    Day: 9/14/2013 Time: 9:28:00
//    Day: 9/14/2013 Time: 9:28 AM
//
//    Day: 5/28/2011 Time: 10:35:00
//    Day: 5/28/2011 Time: 10:35 AM
//
//    Day: 12/25/1979 Time: 14:30:00
//    Day: 12/25/1979 Time: 2:30 PM
```

Remarks

Unlike the Date property. which returns a DateTime value that represents a date without its time component, the TimeOfDay property returns a TimeSpan value that represents a DateTime value's time component.

If you want to display the time of day or retrieve the string representation of the time of day of a DateTime value, you can instead call an overload of the ToString method that has a `format` parameter or use the composite formatting feature with the "t" or "T" standard format string.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TimeSpan

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.Today Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current date.

C#

```
public static DateTime Today { get; }
```

Property Value

DateTime

An object that is set to today's date, with the time component set to 00:00:00.

Examples

The following example uses the Date property to retrieve the current date. It also illustrates how a DateTime value can be formatted using some of the standard date and time format strings. Note that the output produced by the third call to the ToString(String) method uses the g format specifier to include the time component, which is zero.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get the current date.
      DateTime thisDay = DateTime.Today;
      // Display the date in the default (general) format.
      Console.WriteLine(thisDay.ToString());
      Console.WriteLine();
      // Display the date in a variety of formats.
      Console.WriteLine(thisDay.ToString("d"));
```

```
        Console.WriteLine(thisDay.ToString("D"));
        Console.WriteLine(thisDay.ToString("g"));
    }
}
// The example displays output similar to the following:
//    5/3/2012 12:00:00 AM
//
//    5/3/2012
//    Thursday, May 03, 2012
//    5/3/2012 12:00 AM
```

Remarks

Starting with the .NET Framework version 2.0, the return value is a DateTime whose Kind property returns Local.

Because it returns the current date without the current time, the Today property is suitable for use in applications that work with dates only. For details, see Choosing Between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo. In contrast, the TimeOfDay property returns the current time without the current date, and the Now property returns both the current date and the current time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Now
- TimeOfDay

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.UtcNow Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC).

C#

```
public static DateTime UtcNow { get; }
```

Property Value

DateTime

An object whose value is the current UTC date and time.

Examples

The following example uses the SpecifyKind method to demonstrate how the Kind property influences the ToLocalTime and ToUniversalTime conversion methods.

C#

```
// This code example demonstrates the DateTime Kind, Now, and
// UtcNow properties, and the SpecifyKind(), ToLocalTime(),
// and ToUniversalTime() methods.

using System;

class Sample
{
    public static void Main()
    {
        // Get the date and time for the current moment, adjusted
        // to the local time zone.

        DateTime saveNow = DateTime.Now;

        // Get the date and time for the current moment expressed
```

```
        // as coordinated universal time (UTC).

        DateTime saveUtcNow = DateTime.UtcNow;
        DateTime myDt;

        // Display the value and Kind property of the current moment
        // expressed as UTC and local time.

        DisplayNow("UtcNow: ..........", saveUtcNow);
        DisplayNow("Now: .............", saveNow);
        Console.WriteLine();

        // Change the Kind property of the current moment to
        // DateTimeKind.Utc and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Utc);
        Display("Utc: .............", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Local and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Local);
        Display("Local: ...........", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Unspecified and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Unspecified);
        Display("Unspecified: .....", myDt);
    }

    // Display the value and Kind property of a DateTime structure, the
    // DateTime structure converted to local time, and the DateTime
    // structure converted to universal time.

    public static string datePatt = @"M/d/yyyy hh:mm:ss tt";
    public static void Display(string title, DateTime inputDt)
    {
        DateTime dispDt = inputDt;
        string dtString;

        // Display the original DateTime.

        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, dispDt.Kind);

        // Convert inputDt to local time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was universal time.

        dispDt = inputDt.ToLocalTime();
```

```
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToLocalTime:     {0}, Kind = {1}",
                          dtString, dispDt.Kind);

        // Convert inputDt to universal time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was local time.

        dispDt = inputDt.ToUniversalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToUniversalTime: {0}, Kind = {1}",
                          dtString, dispDt.Kind);
        Console.WriteLine();
    }

    // Display the value and Kind property for DateTime.Now and
DateTime.UtcNow.

    public static void DisplayNow(string title, DateTime inputDt)
    {
        string dtString = inputDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, inputDt.Kind);
    }
}

/*
This code example produces the following results:

UtcNow: .......... 5/6/2005 09:34:42 PM, Kind = Utc
Now: ............. 5/6/2005 02:34:42 PM, Kind = Local

Utc: ............. 5/6/2005 02:34:42 PM, Kind = Utc
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 02:34:42 PM, Kind = Utc

Local: ........... 5/6/2005 02:34:42 PM, Kind = Local
  ToLocalTime:     5/6/2005 02:34:42 PM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

Unspecified: ..... 5/6/2005 02:34:42 PM, Kind = Unspecified
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

*/
```

Remarks

The resolution of this property depends on the system timer, which depends on the underlying operating system. It tends to be between 0.5 and 15 milliseconds.

Starting with the .NET Framework version 2.0, the return value is a DateTime whose Kind property returns DateTimeKind.Utc.

An alternative to using UtcNow is DateTimeOffset.UtcNow. While the former indicates that a date and time value is Coordinated Universal Time (UTC) by assigning DateTimeKind.Utc to its Kind property, the latter assigns the date and time value the UTC time's offset (equal to TimeSpan.Zero).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UtcNow
- GetUtcOffset(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Year Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the year component of the date represented by this instance.

C#

```
public int Year { get; }
```

Property Value

Int32

The year, between 1 and 9999.

Examples

The following example demonstrates the Year property.

C#

```
System.DateTime moment = new System.DateTime(
                                1999, 1, 13, 3, 57, 32, 11);
// Year gets 1999.
int year = moment.Year;

// Month gets 1 (January).
int month = moment.Month;

// Day gets 13.
int day = moment.Day;

// Hour gets 3.
int hour = moment.Hour;

// Minute gets 57.
int minute = moment.Minute;

// Second gets 32.
int second = moment.Second;
```

```
// Millisecond gets 11.
int millisecond = moment.Millisecond;
```

Remarks

The Year property returns the year of the current instance in the Gregorian calendar. It does not return the year using the default calendar of the current culture. To retrieve the year using a particular calendar, you can call that calendar's `GetYear` method, as the following code shows.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class YearMethodExample
{
    public static void Main()
    {
      // Initialize date variable and display year
      DateTime date1 = new DateTime(2008, 1, 1, 6, 32, 0);
      Console.WriteLine(date1.Year);                    // Displays 2008

      // Set culture to th-TH
      Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH");
      Console.WriteLine(date1.Year);                    // Displays 2008

      // display year using current culture's calendar
      Calendar thaiCalendar = CultureInfo.CurrentCulture.Calendar;
      Console.WriteLine(thaiCalendar.GetYear(date1));   // Displays 2551

      // display year using Persian calendar
      PersianCalendar persianCalendar = new PersianCalendar();
      Console.WriteLine(persianCalendar.GetYear(date1)); // Displays 1386
   }
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int32
- Calendar

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Add(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance.

```
C#
public DateTime Add (TimeSpan value);
```

Parameters

`value` TimeSpan

A positive or negative time interval.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the time interval represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates the Add method. It calculates the day of the week that is 36 days (864 hours) from this moment.

```
C#
```

```
// Calculate what day of the week is 36 days from this instant.
System.DateTime today = System.DateTime.Now;
System.TimeSpan duration = new System.TimeSpan(36, 0, 0, 0);
System.DateTime answer = today.Add(duration);
System.Console.WriteLine("{0:dddd}", answer);
```

Remarks

You can use the Add method to add more than one kind of time interval (days, hours, minutes, seconds, or milliseconds) in a single operation. This method's behavior is identical to that of the addition operator. The DateTime structure also supports specialized addition methods (such as AddDays, AddHours, and AddMinutes) for each time interval.

The Add method takes into account leap years and the number of days in a month when performing date arithmetic.

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation. The Kind property of the new DateTime instance is the same as that of the current instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TimeSpan
- Add(TimeSpan)
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.AddDays(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of days to the value of this instance.

```
C#
public DateTime AddDays (double value);
```

Parameters

`value` Double

A number of whole and fractional days. The `value` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of days represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example uses the AddDays method to determine the day of the week 36 days after the current date.

```
C#
```

```
using System;

class Class1
{
    static void Main()
    {
        DateTime today = DateTime.Now;
        DateTime answer = today.AddDays(36);
        Console.WriteLine("Today: {0:dddd}", today);
        Console.WriteLine("36 days from today: {0:dddd}", answer);
    }
}
// The example displays output like the following:
//       Today: Wednesday
//       36 days from today: Thursday
```

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

The fractional part of `value` is the fractional part of a day. For example, 4.5 is equivalent to 4 days, 12 hours, 0 minutes, 0 seconds, 0 milliseconds, and 0 ticks.

In .NET 6 and earlier versions, the `value` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `value` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

The AddDays method takes into account leap years and the number of days in a month when performing date arithmetic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddHours(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of hours to the value of this instance.

```
C#

public DateTime AddHours (double value);
```

Parameters

`value` Double

A number of whole and fractional hours. The `value` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of hours represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example uses the AddHours method to add a number of whole and fractional values to a date and time. It also illustrates the loss of precision caused by passing the method a value that includes a fractional component.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double[] hours = {.08333, .16667, .25, .33333, .5, .66667, 1, 2,
                        29, 30, 31, 90, 365};
      DateTime dateValue = new DateTime(2009, 3, 1, 12, 0, 0);

      foreach (double hour in hours)
         Console.WriteLine("{0} + {1} hour(s) = {2}", dateValue, hour,
                           dateValue.AddHours(hour));
   }
}
// The example displays the following output on a system whose current
// culture is en-US:
//    3/1/2009 12:00:00 PM + 0.08333 hour(s) = 3/1/2009 12:04:59 PM
//    3/1/2009 12:00:00 PM + 0.16667 hour(s) = 3/1/2009 12:10:00 PM
//    3/1/2009 12:00:00 PM + 0.25 hour(s) = 3/1/2009 12:15:00 PM
//    3/1/2009 12:00:00 PM + 0.33333 hour(s) = 3/1/2009 12:19:59 PM
//    3/1/2009 12:00:00 PM + 0.5 hour(s) = 3/1/2009 12:30:00 PM
//    3/1/2009 12:00:00 PM + 0.66667 hour(s) = 3/1/2009 12:40:00 PM
//    3/1/2009 12:00:00 PM + 1 hour(s) = 3/1/2009 1:00:00 PM
//    3/1/2009 12:00:00 PM + 2 hour(s) = 3/1/2009 2:00:00 PM
//    3/1/2009 12:00:00 PM + 29 hour(s) = 3/2/2009 5:00:00 PM
//    3/1/2009 12:00:00 PM + 30 hour(s) = 3/2/2009 6:00:00 PM
//    3/1/2009 12:00:00 PM + 31 hour(s) = 3/2/2009 7:00:00 PM
//    3/1/2009 12:00:00 PM + 90 hour(s) = 3/5/2009 6:00:00 AM
//    3/1/2009 12:00:00 PM + 365 hour(s) = 3/16/2009 5:00:00 PM
```

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation. The Kind property of the returned DateTime object is the same as that of `value`.

The fractional part of `value` is the fractional part of an hour. For example, 4.5 is equivalent to 4 hours, 30 minutes, 0 seconds, 0 milliseconds, and 0 ticks.

In .NET 6 and earlier versions, the `value` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `value` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

Converting time intervals of less than an hour to a fraction can involve a loss of precision if the result is a non-terminating repeating decimal. (For example, one minute is 0.016667 of an hour.) If this is problematic, you can use the Add method, which enables you to specify more than one kind of time interval in a single method call and eliminates the need to convert time intervals to fractional parts of an hour.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(TimeSpan)
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddMicroseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of microseconds to the value of this instance.

```
C#

public DateTime AddMicroseconds (double value);
```

Parameters

`value` Double

A number of whole and fractional microseconds. The `value` parameter can be negative or positive. Note that this value is rounded to the nearest integer.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of microseconds represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than MinValue or greater than MaxValue.

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

The fractional part of value is the fractional part of a microsecond. For example, 4.5 is equivalent to 4 microseconds and 50 ticks, where one microsecond = 10 ticks.

The value parameter is rounded to the nearest integer.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddMilliseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of milliseconds to the value of this instance.

```
C#

public DateTime AddMilliseconds (double value);
```

Parameters

`value` Double

A number of whole and fractional milliseconds. The `value` parameter can be negative or positive. Note that this value is rounded to the nearest integer.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of milliseconds represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example uses the AddMilliseconds method to add one millisecond and 1.5 milliseconds to a DateTime value. It then displays each new value and displays the

difference between it and the original value. The difference is displayed both as a time span and as a number of ticks. The example makes it clear that one millisecond equals 10,000 ticks. It also shows that fractional milliseconds are rounded before performing the addition; the DateTime value that results from adding 1.5 milliseconds to the original date is 2 milliseconds greater than the original date.

C#

```
string dateFormat = "MM/dd/yyyy hh:mm:ss.fffffff";
DateTime date1 = new DateTime(2010, 9, 8, 16, 0, 0);
Console.WriteLine("Original date: {0} ({1:N0} ticks)\n",
                  date1.ToString(dateFormat), date1.Ticks);

DateTime date2 = date1.AddMilliseconds(1);
Console.WriteLine("Second date:   {0} ({1:N0} ticks)",
                  date2.ToString(dateFormat), date2.Ticks);
Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)\n",
                  date2 - date1, date2.Ticks - date1.Ticks);

DateTime date3 = date1.AddMilliseconds(1.5);
Console.WriteLine("Third date:    {0} ({1:N0} ticks)",
                  date3.ToString(dateFormat), date3.Ticks);
Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)",
                  date3 - date1, date3.Ticks - date1.Ticks);
// The example displays the following output:
//    Original date: 09/08/2010 04:00:00.0000000 (634,195,584,000,000,000
ticks)
//
//    Second date:   09/08/2010 04:00:00.0010000 (634,195,584,000,010,000
ticks)
//    Difference between dates: 00:00:00.0010000 (10,000 ticks)
//
//    Third date:    09/08/2010 04:00:00.0020000 (634,195,584,000,020,000
ticks)
//    Difference between dates: 00:00:00.0020000 (20,000 ticks)
```

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

The fractional part of `value` is the fractional part of a millisecond. For example, 4.5 is equivalent to 4 milliseconds and 5000 ticks, where one millisecond = 10000 ticks.

In .NET 6 and earlier versions, the `value` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `value`

parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddMinutes(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of minutes to the value of this instance.

```
C#

public DateTime AddMinutes (double value);
```

Parameters

`value` Double

A number of whole and fractional minutes. The `value` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of minutes represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example uses the AddMinutes method to add a number of whole and fractional values to a date and time.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime dateValue = new DateTime(2013, 9, 15, 12, 0, 0);

      double[] minutes = { .01667, .08333, .16667, .25, .33333,
                           .5, .66667, 1, 2, 15, 30, 17, 45,
                           60, 180, 60 * 24 };

      foreach (double minute in minutes)
         Console.WriteLine("{0} + {1} minute(s) = {2}", dateValue, minute,
                           dateValue.AddMinutes(minute));
   }
}
// The example displays the following output on a system whose current
culture is en-US:
//    9/15/2013 12:00:00 PM + 0.01667 minute(s) = 9/15/2013 12:00:01 PM
//    9/15/2013 12:00:00 PM + 0.08333 minute(s) = 9/15/2013 12:00:05 PM
//    9/15/2013 12:00:00 PM + 0.16667 minute(s) = 9/15/2013 12:00:10 PM
//    9/15/2013 12:00:00 PM + 0.25 minute(s) = 9/15/2013 12:00:15 PM
//    9/15/2013 12:00:00 PM + 0.33333 minute(s) = 9/15/2013 12:00:20 PM
//    9/15/2013 12:00:00 PM + 0.5 minute(s) = 9/15/2013 12:00:30 PM
//    9/15/2013 12:00:00 PM + 0.66667 minute(s) = 9/15/2013 12:00:40 PM
//    9/15/2013 12:00:00 PM + 1 minute(s) = 9/15/2013 12:01:00 PM
//    9/15/2013 12:00:00 PM + 2 minute(s) = 9/15/2013 12:02:00 PM
//    9/15/2013 12:00:00 PM + 15 minute(s) = 9/15/2013 12:15:00 PM
//    9/15/2013 12:00:00 PM + 30 minute(s) = 9/15/2013 12:30:00 PM
//    9/15/2013 12:00:00 PM + 17 minute(s) = 9/15/2013 12:17:00 PM
//    9/15/2013 12:00:00 PM + 45 minute(s) = 9/15/2013 12:45:00 PM
//    9/15/2013 12:00:00 PM + 60 minute(s) = 9/15/2013 1:00:00 PM
//    9/15/2013 12:00:00 PM + 180 minute(s) = 9/15/2013 3:00:00 PM
//    9/15/2013 12:00:00 PM + 1440 minute(s) = 9/16/2013 12:00:00 PM
```

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

The fractional part of `value` is the fractional part of a minute. For example, 4.5 is equivalent to 4 minutes, 30 seconds, 0 milliseconds, and 0 ticks.

In .NET 6 and earlier versions, the `value` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `value` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddMonths(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of months to the value of this instance.

```
C#
public DateTime AddMonths (int months);
```

Parameters

`months` Int32

A number of months. The `months` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and `months`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

-or-

`months` is less than -120,000 or greater than 120,000.

Examples

The following example adds between zero and fifteen months to the last day of December, 2015. In this case, the AddMonths method returns the date of the last day of each month, and successfully handles leap years.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      var dat = new DateTime(2015, 12, 31);
      for (int ctr = 0; ctr <= 15; ctr++)
         Console.WriteLine(dat.AddMonths(ctr).ToString("d"));
   }
}
// The example displays the following output:
//       12/31/2015
//       1/31/2016
//       2/29/2016
//       3/31/2016
//       4/30/2016
//       5/31/2016
//       6/30/2016
//       7/31/2016
//       8/31/2016
//       9/30/2016
//       10/31/2016
//       11/30/2016
//       12/31/2016
//       1/31/2017
//       2/28/2017
//       3/31/2017
```

Remarks

This method does not change the value of this DateTime object. Instead, it returns a new DateTime object whose value is the result of this operation.

The AddMonths method calculates the resulting month and year, taking into account leap years and the number of days in a month, then adjusts the day part of the resulting DateTime object. If the resulting day is not a valid day in the resulting month, the last valid day of the resulting month is used. For example, March 31st + 1 month = April 30th, and March 31st - 1 month = February 28 for a non-leap year and February 29 for a leap year.

The time-of-day part of the resulting DateTime object remains the same as this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddSeconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of seconds to the value of this instance.

```
C#

public DateTime AddSeconds (double value);
```

Parameters

`value` Double

A number of whole and fractional seconds. The `value` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of seconds represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example uses the AddSeconds method to add 30 seconds and the number of seconds in one day to a DateTime value. It then displays each new value and displays the difference between it and the original value. The difference is displayed both as a time span and as a number of ticks.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string dateFormat = "MM/dd/yyyy hh:mm:ss";
      DateTime date1 = new DateTime(2014, 9, 8, 16, 0, 0);
      Console.WriteLine("Original date: {0} ({1:N0} ticks)\n",
                        date1.ToString(dateFormat), date1.Ticks);

      DateTime date2 = date1.AddSeconds(30);
      Console.WriteLine("Second date:   {0} ({1:N0} ticks)",
                        date2.ToString(dateFormat), date2.Ticks);
      Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)\n",
                        date2 - date1, date2.Ticks - date1.Ticks);

      // Add 1 day's worth of seconds (60 secs. * 60 mins * 24 hrs.
      DateTime date3 = date1.AddSeconds(60 * 60 * 24);
      Console.WriteLine("Third date:    {0} ({1:N0} ticks)",
                        date3.ToString(dateFormat), date3.Ticks);
      Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)",
                        date3 - date1, date3.Ticks - date1.Ticks);
   }
}
// The example displays the following output:
//    Original date: 09/08/2014 04:00:00 (635,457,888,000,000,000 ticks)
//
//    Second date:   09/08/2014 04:00:30 (635,457,888,300,000,000 ticks)
//    Difference between dates: 00:00:30 (300,000,000 ticks)
//
//    Third date:    09/09/2014 04:00:00 (635,458,752,000,000,000 ticks)
//    Difference between dates: 1.00:00:00 (864,000,000,000 ticks)
```

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

The fractional part of `value` is the fractional part of a second. For example, 4.5 is equivalent to 4 seconds, 500 milliseconds, and 0 ticks.

In .NET 6 and earlier versions, the `value` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `value` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddTicks(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of ticks to the value of this instance.

```
C#
public DateTime AddTicks (long value);
```

Parameters

`value` Int64

A number of 100-nanosecond ticks. The `value` parameter can be positive or negative.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the time represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Remarks

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- Ticks
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.AddYears(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTime that adds the specified number of years to the value of this instance.

```
C#

public DateTime AddYears (int value);
```

Parameters

`value` Int32

A number of years. The `value` parameter can be negative or positive.

Returns

DateTime

An object whose value is the sum of the date and time represented by this instance and the number of years represented by `value`.

Exceptions

ArgumentOutOfRangeException

`value` or the resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Remarks

This method does not change the value of this DateTime object. Instead, it returns a new DateTime object whose value is the result of this operation.

The AddYears method calculates the resulting year taking into account leap years. The month and time-of-day part of the resulting DateTime object remains the same as this

instance.

If the current instance represents the leap day in a leap year, the return value depends on the target date:

- If `value` + DateTime.Year is also a leap year, the return value represents the leap day in that year. For example, if four years is added to February 29, 2012, the date returned is February 29, 2016.
- If `value` + DateTime.Year is not a leap year, the return value represents the day before the leap day in that year. For example, if one year is added to February 29, 2012, the date returned is February 28, 2013.

The following example illustrates using the AddYears method with a DateTime value that represents a leap year day. It displays the date for the fifteen years prior to and the fifteen years that follow February 29, 2000.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime baseDate = new DateTime(2000, 2, 29);
      Console.WriteLine("    Base Date:        {0:d}\n", baseDate);

      // Show dates of previous fifteen years.
      for (int ctr = -1; ctr >= -15; ctr--)
         Console.WriteLine("{0,2} year(s) ago:        {1:d}",
                           Math.Abs(ctr), baseDate.AddYears(ctr));
      Console.WriteLine();

      // Show dates of next fifteen years.
      for (int ctr = 1; ctr <= 15; ctr++)
         Console.WriteLine("{0,2} year(s) from now:   {1:d}",
                           ctr, baseDate.AddYears(ctr));
   }
}
// The example displays the following output:
//           Base Date:        2/29/2000
//
//        1 year(s) ago:        2/28/1999
//        2 year(s) ago:        2/28/1998
//        3 year(s) ago:        2/28/1997
//        4 year(s) ago:        2/29/1996
//        5 year(s) ago:        2/28/1995
//        6 year(s) ago:        2/28/1994
//        7 year(s) ago:        2/28/1993
//        8 year(s) ago:        2/29/1992
```

```
//        9 year(s) ago:        2/28/1991
//       10 year(s) ago:        2/28/1990
//       11 year(s) ago:        2/28/1989
//       12 year(s) ago:        2/29/1988
//       13 year(s) ago:        2/28/1987
//       14 year(s) ago:        2/28/1986
//       15 year(s) ago:        2/28/1985
//
//        1 year(s) from now:   2/28/2001
//        2 year(s) from now:   2/28/2002
//        3 year(s) from now:   2/28/2003
//        4 year(s) from now:   2/29/2004
//        5 year(s) from now:   2/28/2005
//        6 year(s) from now:   2/28/2006
//        7 year(s) from now:   2/28/2007
//        8 year(s) from now:   2/29/2008
//        9 year(s) from now:   2/28/2009
//       10 year(s) from now:   2/28/2010
//       11 year(s) from now:   2/28/2011
//       12 year(s) from now:   2/29/2012
//       13 year(s) from now:   2/28/2013
//       14 year(s) from now:   2/28/2014
//       15 year(s) from now:   2/28/2015
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Compare(DateTime, DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance.

C#

```
public static int Compare (DateTime t1, DateTime t2);
```

Parameters

t1 DateTime

The first object to compare.

t2 DateTime

The second object to compare.

Returns

Int32

A signed number indicating the relative values of `t1` and `t2`.

Expand table

Value Type	Condition
Less than zero	`t1` is earlier than `t2`.
Zero	`t1` is the same as `t2`.
Greater than zero	`t1` is later than `t2`.

Examples

The following example demonstrates the Compare method.

C#

```
DateTime date1 = new DateTime(2009, 8, 1, 0, 0, 0);
DateTime date2 = new DateTime(2009, 8, 1, 12, 0, 0);
int result = DateTime.Compare(date1, date2);
string relationship;

if (result < 0)
   relationship = "is earlier than";
else if (result == 0)
   relationship = "is the same time as";
else
   relationship = "is later than";

Console.WriteLine("{0} {1} {2}", date1, relationship, date2);

// The example displays the following output for en-us culture:
//    8/1/2009 12:00:00 AM is earlier than 8/1/2009 12:00:00 PM
```

Remarks

To determine the relationship of `t1` to `t2`, the Compare method compares the Ticks property of `t1` and `t2` but ignores their Kind property. Before comparing DateTime objects, ensure that the objects represent times in the same time zone.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(Object)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the value of this instance to a specified DateTime value and indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.

Overloads

Expand table

CompareTo(DateTime)	Compares the value of this instance to a specified DateTime value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.
CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified DateTime value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.

Remarks

The two overloads of the CompareTo method return a signed number that indicates the relative value of this instance and the `value` argument, as shown in the following table.

Expand table

Value	Description
Less than zero	This instance is earlier than `value`.
Zero	This instance is the same as `value`.
Greater than zero	This instance is later than `value`.

CompareTo(DateTime)

Compares the value of this instance to a specified DateTime value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.

```
C#

public int CompareTo (DateTime value);
```

Parameters

`value` DateTime

The object to compare to the current instance.

Returns

Int32

A signed number indicating the relative values of this instance and the `value` parameter.

Expand table

Value	Description
Less than zero	This instance is earlier than `value`.
Zero	This instance is the same as `value`.
Greater than zero	This instance is later than `value`.

Implements

CompareTo(T)

Examples

The following example instantiates three DateTime objects, one that represents today's date, another that represents the date one year previously, and a third that represents the date one year in the future. It then calls the CompareTo(DateTime) method and displays the result of the comparison.

```
C#
```

```
using System;

public class DateTimeComparison
{
   private enum DateComparisonResult
   {
      Earlier = -1,
      Later = 1,
      TheSame = 0
   };

   public static void Main()
   {
      DateTime thisDate = DateTime.Today;

      // Define two DateTime objects for today's date
      // next year and last year
      DateTime thisDateNextYear, thisDateLastYear;

      // Call AddYears instance method to add/substract 1 year
      thisDateNextYear = thisDate.AddYears(1);
      thisDateLastYear = thisDate.AddYears(-1);

      // Compare dates
      //
      DateComparisonResult comparison;
      // Compare today to last year
      comparison = (DateComparisonResult)
thisDate.CompareTo(thisDateLastYear);
      Console.WriteLine("CompareTo method returns {0}: {1:d} is {2} than
{3:d}",
                        (int) comparison, thisDate,
comparison.ToString().ToLower(),
                        thisDateLastYear);

      // Compare today to next year
      comparison = (DateComparisonResult)
thisDate.CompareTo(thisDateNextYear);
      Console.WriteLine("CompareTo method returns {0}: {1:d} is {2} than
{3:d}",
                        (int) comparison, thisDate,
comparison.ToString().ToLower(),
                        thisDateNextYear);
   }
}
//
// If run on October 20, 2006, the example produces the following output:
//    CompareTo method returns 1: 10/20/2006 is later than 10/20/2005
//    CompareTo method returns -1: 10/20/2006 is earlier than 10/20/2007
```

Remarks

To determine the relationship of the current instance to `value`, the CompareTo method compares the Ticks property of the current instance and `value` but ignores their Kind property. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind properties.

This method implements the System.IComparable<T> interface and performs slightly better than the DateTime.CompareTo(Object) method overload because it does not have to convert the `value` parameter to an object.

See also

- Compare(DateTime, DateTime)
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares the value of this instance to a specified object that contains a specified DateTime value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

A boxed object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Value	Description
Less than zero	This instance is earlier than `value`.
Zero	This instance is the same as `value`.
Greater than zero	This instance is later than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a DateTime.

Examples

The following example demonstrates the CompareTo method.

C#

```
System.DateTime theDay = new System.DateTime(System.DateTime.Today.Year,
7, 28);
int compareValue;

try
{
    compareValue = theDay.CompareTo(DateTime.Today);
}
catch (ArgumentException)
{
   Console.WriteLine("Value is not a DateTime");
   return;
```

```
}

if (compareValue < 0)
   System.Console.WriteLine("{0:d} is in the past.", theDay);
else if (compareValue == 0)
   System.Console.WriteLine("{0:d} is today!", theDay);
else // compareValue > 0
   System.Console.WriteLine("{0:d} has not come yet.", theDay);
```

Remarks

To determine the relationship of the current instance to `value`, the CompareTo method compares the Ticks property of the current instance and `value` but ignores their Kind property. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind properties.

Any instance of DateTime, regardless of its value, is considered greater than `null`.

See also

- Compare(DateTime, DateTime)
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTime.DaysInMonth(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of days in the specified month and year.

```
C#
public static int DaysInMonth (int year, int month);
```

Parameters

`year` Int32

The year.

`month` Int32

The month (a number ranging from 1 to 12).

Returns

Int32

The number of days in `month` for the specified `year`.

For example, if `month` equals 2 for February, the return value is 28 or 29 depending upon whether `year` is a leap year.

Exceptions

ArgumentOutOfRangeException

`month` is less than 1 or greater than 12.

-or-

`year` is less than 1 or greater than 9999.

Examples

The following example demonstrates how to use the DaysInMonth method to determine the number of days in July 2001, February 1998 (a non-leap year), and February 1996 (a leap year).

C#

```
using System;

class Example
{
    static void Main()
    {
        const int July = 7;
        const int Feb = 2;

        int daysInJuly = System.DateTime.DaysInMonth(2001, July);
        Console.WriteLine(daysInJuly);

        // daysInFeb gets 28 because the year 1998 was not a leap year.
        int daysInFeb = System.DateTime.DaysInMonth(1998, Feb);
        Console.WriteLine(daysInFeb);

        // daysInFebLeap gets 29 because the year 1996 was a leap year.
        int daysInFebLeap = System.DateTime.DaysInMonth(1996, Feb);
        Console.WriteLine(daysInFebLeap);
    }
}
// The example displays the following output:
//       31
//       28
//       29
```

The following example displays the number of days in each month of a year specified in an integer array.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      int[] years = { 2012, 2014 };
      DateTimeFormatInfo dtfi = DateTimeFormatInfo.CurrentInfo;
      Console.WriteLine("Days in the Month for the {0} culture " +
                        "using the {1} calendar\n",
                        CultureInfo.CurrentCulture.Name,
```

```
                        dtfi.Calendar.GetType().Name.Replace("Calendar",
""));
      Console.WriteLine("{0,-10}{1,-15}{2,4}\n", "Year", "Month", "Days");

      foreach (var year in years) {
         for (int ctr = 0; ctr <= dtfi.MonthNames.Length - 1; ctr++) {
            if (String.IsNullOrEmpty(dtfi.MonthNames[ctr]))
               continue;

            Console.WriteLine("{0,-10}{1,-15}{2,4}", year,
                              dtfi.MonthNames[ctr],
                              DateTime.DaysInMonth(year, ctr + 1));
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    Days in the Month for the en-US culture using the Gregorian calendar
//
//    Year      Month          Days
//
//    2012      January          31
//    2012      February         29
//    2012      March            31
//    2012      April            30
//    2012      May              31
//    2012      June             30
//    2012      July             31
//    2012      August           31
//    2012      September        30
//    2012      October          31
//    2012      November         30
//    2012      December         31
//
//    2014      January          31
//    2014      February         28
//    2014      March            31
//    2014      April            30
//    2014      May              31
//    2014      June             30
//    2014      July             31
//    2014      August           31
//    2014      September        30
//    2014      October          31
//    2014      November         30
//    2014      December         31
```

Remarks

The DaysInMonth method always interprets `month` and `year` as the month and year of the Gregorian calendar even if the Gregorian calendar is not the current culture's current

calendar. To get the number of days in a specified month of a particular calendar, call that calendar's GetDaysInMonth method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Deconstruct Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Deconstruct(DateOnly, TimeOnly)	Deconstructs this DateTime instance by DateOnly and TimeOnly.
Deconstruct(Int32, Int32, Int32)	Deconstructs this DateOnly instance by Year, Month, and Day.

Deconstruct(DateOnly, TimeOnly)

Deconstructs this DateTime instance by DateOnly and TimeOnly.

C#

```
public void Deconstruct (out DateOnly date, out TimeOnly time);
```

Parameters

`date` DateOnly

When this method returns, represents the DateOnly value of this DateTime instance.

`time` TimeOnly

When this method returns, represents the TimeOnly value of this DateTime instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Deconstruct(Int32, Int32, Int32)

Deconstructs this DateOnly instance by Year, Month, and Day.

```
C#
public void Deconstruct (out int year, out int month, out int day);
```

Parameters

`year` Int32

When this method returns, represents the Year value of this DateTime instance.

`month` Int32

When this method returns, represents the Month value of this DateTime instance.

`day` Int32

When this method returns, represents the Day value of this DateTime instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTime.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two DateTime objects, or a DateTime instance and another object or DateTime, have the same value.

Overloads

Expand table

Equals(DateTime)	Returns a value indicating whether the value of this instance is equal to the value of the specified DateTime instance.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(DateTime, DateTime)	Returns a value indicating whether two DateTime instances have the same date and time value.

Equals(DateTime)

Returns a value indicating whether the value of this instance is equal to the value of the specified DateTime instance.

C#

```
public bool Equals (DateTime value);
```

Parameters

`value` DateTime

The object to compare to this instance.

Returns

Boolean

`true` if the `value` parameter equals the value of this instance; otherwise, `false`.

Implements

Equals(T)

Examples

The following example demonstrates the Equals method.

C#

```
using System;

public class Application
{
    public static void Main()
    {
        // Create some DateTime objects.
        DateTime one = DateTime.UtcNow;

        DateTime two = DateTime.Now;

        DateTime three = one;

        // Compare the DateTime objects and display the results.
        bool result = one.Equals(two);

        Console.WriteLine("The result of comparing DateTime object one
and two is: {0}.", result);

        result = one.Equals(three);

        Console.WriteLine("The result of comparing DateTime object one
and three is: {0}.", result);
    }
}

// This code example displays the following:
//
// The result of comparing DateTime object one and two is: False.
// The result of comparing DateTime object one and three is: True.
```

Remarks

The current instance and `value` are equal if their Ticks property values are equal. Their Kind property values are not considered in the test for equality.

This method implements the System.IEquatable<T> interface, and performs slightly better than the Equals method because the `value` parameter does not have to be converted to an object.

See also

- Equals(Object)
- Compare(DateTime, DateTime)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? value);
```

Parameters

`value` Object

The object to compare to this instance.

Returns

Boolean

`true` if `value` is an instance of DateTime and equals the value of this instance; otherwise, `false`.

Examples

The following example demonstrates the Equals method.

C#

```
using System;

public class Application
{
    public static void Main()
    {
        // Create some DateTime objects.
        DateTime one = DateTime.UtcNow;

        DateTime two = DateTime.Now;

        DateTime three = one;

        // Compare the DateTime objects and display the results.
        bool result = one.Equals(two);

        Console.WriteLine("The result of comparing DateTime object one
and two is: {0}.", result);

        result = one.Equals(three);

        Console.WriteLine("The result of comparing DateTime object one
and three is: {0}.", result);
    }
}

// This code example displays the following:
//
// The result of comparing DateTime object one and two is: False.
// The result of comparing DateTime object one and three is: True.
```

Remarks

The current instance and `value` are equal if their Ticks property values are equal. Their Kind property values are not considered in the test for equality.

See also

- Compare(DateTime, DateTime)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(DateTime, DateTime)

Returns a value indicating whether two DateTime instances have the same date and time value.

C#

```
public static bool Equals (DateTime t1, DateTime t2);
```

Parameters

`t1` DateTime

The first object to compare.

`t2` DateTime

The second object to compare.

Returns

Boolean

`true` if the two values are equal; otherwise, `false`.

Examples

The following example demonstrates the Equals method.

```C#
System.DateTime today1 =
        new System.DateTime(System.DateTime.Today.Ticks);
System.DateTime today2 =
        new System.DateTime(System.DateTime.Today.Ticks);
System.DateTime tomorrow =
        new System.DateTime(
                    System.DateTime.Today.AddDays(1).Ticks);

// todayEqualsToday gets true.
bool todayEqualsToday = System.DateTime.Equals(today1, today2);

// todayEqualsTomorrow gets false.
bool todayEqualsTomorrow = System.DateTime.Equals(today1, tomorrow);
```

Remarks

`t1` and `t2` are equal if their Ticks property values are equal. Their Kind property values are not considered in the test for equality.

See also

- Boolean
- Compare(DateTime, DateTime)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.FromBinary(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Deserializes a 64-bit binary value and recreates an original serialized DateTime object.

C#

```
public static DateTime FromBinary (long dateData);
```

Parameters

dateData Int64

A 64-bit signed integer that encodes the Kind property in a 2-bit field and the Ticks property in a 62-bit field.

Returns

DateTime

An object that is equivalent to the DateTime object that was serialized by the ToBinary() method.

Exceptions

ArgumentException

`dateData` is less than DateTime.MinValue or greater than DateTime.MaxValue.

Remarks

For more information about this API, see Supplemental API remarks for DateTime.FromBinary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToBinary()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.FromFileTime(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified Windows file time to an equivalent local time.

C#

```
public static DateTime FromFileTime (long fileTime);
```

Parameters

`fileTime` Int64

A Windows file time expressed in ticks.

Returns

DateTime

An object that represents the local time equivalent of the date and time represented by the `fileTime` parameter.

Exceptions

ArgumentOutOfRangeException

`fileTime` is less than 0 or represents a time greater than DateTime.MaxValue.

Examples

The following example demonstrates the FromFileTime method.

C#

```
public System.TimeSpan FileAge(long fileCreationTime) {

    System.DateTime now = System.DateTime.Now;
    try {
```

```
        System.DateTime fCreationTime =
            System.DateTime.FromFileTime(fileCreationTime);
        System.TimeSpan fileAge = now.Subtract(fCreationTime);
        return fileAge;
    }
    catch (ArgumentOutOfRangeException) {
        // fileCreationTime is not valid, so re-throw the exception.
        throw;
    }
}
```

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

The `fileTime` parameter specifies a file time expressed in 100-nanosecond ticks.

Starting with the .NET Framework version 2.0, the return value is a DateTime whose Kind property is DateTimeKind.Local.

Notes to Callers

Ordinarily, the FromFileTime(Int64) method restores a DateTime value that was saved by the ToFileTime() method. However, the two values may differ under the following conditions:

- If the serialization and deserialization of the DateTime value occur in different time zones. For example, if a DateTime value with a time of 12:30 P.M. in the U.S. Eastern Time zone is serialized, and then deserialized in the U.S. Pacific Time zone, the original value of 12:30 P.M. is adjusted to 9:30 A.M. to reflect the difference between the two time zones.

- If the DateTime value that is serialized represents an invalid time in the local time zone. In this case, the ToFileTime() method adjusts the restored DateTime value so that it represents a valid time in the local time zone.

  For example, the transition from standard time to daylight saving time occurs in the U.S. Pacific Time zone on March 14, 2010, at 2:00 A.M., when the time advances by one hour, to 3:00 A.M. This hour interval is an invalid time, that is, a time interval that does not exist in this time zone. The following example shows that

when a time that falls within this range is converted to a long integer value by the ToFileTime() method and is then restored by the FromFileTime(Int64) method, the original value is adjusted to become a valid time. You can determine whether a particular date and time value may be subject to modification by passing it to the IsInvalidTime(DateTime) method, as the example illustrates.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime date1 = new DateTime(2010, 3, 14, 2, 30, 00);
      Console.WriteLine("Invalid Time: {0}",
                        TimeZoneInfo.Local.IsInvalidTime(date1));
      long ft = date1.ToFileTime();
      DateTime date2 = DateTime.FromFileTime(ft);
      Console.WriteLine("{0} -> {1}", date1, date2);
   }
}
// The example displays the following output:
//       Invalid Time: True
//       3/14/2010 2:30:00 AM -> 3/14/2010 3:30:00 AM
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToUniversalTime()
- GetUtcOffset(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.FromFileTimeUtc(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified Windows file time to an equivalent UTC time.

```
C#
public static DateTime FromFileTimeUtc (long fileTime);
```

Parameters

`fileTime` Int64

A Windows file time expressed in ticks.

Returns

DateTime

An object that represents the UTC time equivalent of the date and time represented by the `fileTime` parameter.

Exceptions

ArgumentOutOfRangeException

`fileTime` is less than 0 or represents a time greater than DateTime.MaxValue.

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

The `fileTime` parameter specifies a file time expressed in 100-nanosecond ticks.

Starting with the .NET Framework version 2.0, the return value is a DateTime whose Kind property is Utc.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToUniversalTime()
- GetUtcOffset(DateTime)
- FromFileTime(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.FromOADate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a DateTime equivalent to the specified OLE Automation Date.

```
C#
public static DateTime FromOADate (double d);
```

Parameters

d Double

An OLE Automation Date value.

Returns

DateTime

An object that represents the same date and time as `d`.

Exceptions

ArgumentException

The date is not a valid OLE Automation Date value.

Remarks

The `d` parameter is a double-precision floating-point number that represents a date as the number of days before or after the base date, midnight, 30 December 1899. The sign and integral part of `d` encode the date as a positive or negative day displacement from 30 December 1899, and the absolute value of the fractional part of `d` encodes the time of day as a fraction of a day displacement from midnight. `d` must be a value between negative 657435.0 through positive 2958465.99999999.

Note that because of the way dates are encoded, there are two ways of representing any time of day on 30 December 1899. For example, -0.5 and 0.5 both mean noon on 30 December 1899 because a day displacement of plus or minus zero days from the base date is still the base date, and a half day displacement from midnight is noon.

See ToOADate for more information on OLE Automation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ToOADate()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.GetDateTimeFormats Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.

Overloads

Expand table

GetDateTimeFormats()	Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.
GetDateTimeFormats(Char)	Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier.
GetDateTimeFormats(IFormatProvider)	Converts the value of this instance to all the string representations supported by the standard date and time format specifiers and the specified culture-specific formatting information.
GetDateTimeFormats(Char, IFormatProvider)	Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier and culture-specific formatting information.

Remarks

Important

You should not assume that multiple calls to the **GetDateTimeFormats** overloads will return identical data. Depending on the specific overload, the data returned by

this method can change if the current culture changes, the user overrides individual cultural settings, or an update occurs to the system's cultural data.

GetDateTimeFormats()

Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.

C#

```
public string[] GetDateTimeFormats ();
```

Returns

String[]

A string array where each element is the representation of the value of this instance formatted with one of the standard date and time format specifiers.

Examples

The following example demonstrates the DateTime.GetDateTimeFormats() method. It displays the string representation of a date using all possible standard date and time formats in the computer's current culture, which in this case is en-US.

C#

```
DateTime july28 = new DateTime(2009, 7, 28, 5, 23, 15, 16);

string[] july28Formats = july28.GetDateTimeFormats();

// Print out july28 in all DateTime formats using the default culture.
foreach (string format in july28Formats) {
    Console.WriteLine(format);
}
```

The example displays the following output:

```
7/28/2009
7/28/09
07/28/09
```

```
07/28/2009
09/07/28
2009-07-28
28-Jul-09
Tuesday, July 28, 2009
July 28, 2009
Tuesday, 28 July, 2009
28 July, 2009
Tuesday, July 28, 2009 5:23 AM
Tuesday, July 28, 2009 05:23 AM
Tuesday, July 28, 2009 5:23
Tuesday, July 28, 2009 05:23
July 28, 2009 5:23 AM
July 28, 2009 05:23 AM
July 28, 2009 5:23
July 28, 2009 05:23
Tuesday, 28 July, 2009 5:23 AM
Tuesday, 28 July, 2009 05:23 AM
Tuesday, 28 July, 2009 5:23
Tuesday, 28 July, 2009 05:23
28 July, 2009 5:23 AM
28 July, 2009 05:23 AM
28 July, 2009 5:23
28 July, 2009 05:23
Tuesday, July 28, 2009 5:23:15 AM
Tuesday, July 28, 2009 05:23:15 AM
Tuesday, July 28, 2009 5:23:15
Tuesday, July 28, 2009 05:23:15
July 28, 2009 5:23:15 AM
July 28, 2009 05:23:15 AM
July 28, 2009 5:23:15
July 28, 2009 05:23:15
Tuesday, 28 July, 2009 5:23:15 AM
Tuesday, 28 July, 2009 05:23:15 AM
Tuesday, 28 July, 2009 5:23:15
Tuesday, 28 July, 2009 05:23:15
28 July, 2009 5:23:15 AM
28 July, 2009 05:23:15 AM
28 July, 2009 5:23:15
28 July, 2009 05:23:15
7/28/2009 5:23 AM
7/28/2009 05:23 AM
7/28/2009 5:23
7/28/2009 05:23
7/28/09 5:23 AM
7/28/09 05:23 AM
7/28/09 5:23
7/28/09 05:23
07/28/09 5:23 AM
07/28/09 05:23 AM
07/28/09 5:23
07/28/09 05:23
07/28/2009 5:23 AM
07/28/2009 05:23 AM
07/28/2009 5:23
```

```
07/28/2009 05:23
09/07/28 5:23 AM
09/07/28 05:23 AM
09/07/28 5:23
09/07/28 05:23
2009-07-28 5:23 AM
2009-07-28 05:23 AM
2009-07-28 5:23
2009-07-28 05:23
28-Jul-09 5:23 AM
28-Jul-09 05:23 AM
28-Jul-09 5:23
28-Jul-09 05:23
7/28/2009 5:23:15 AM
7/28/2009 05:23:15 AM
7/28/2009 5:23:15
7/28/2009 05:23:15
7/28/09 5:23:15 AM
7/28/09 05:23:15 AM
7/28/09 5:23:15
7/28/09 05:23:15
07/28/09 5:23:15 AM
07/28/09 05:23:15 AM
07/28/09 5:23:15
07/28/09 05:23:15
07/28/2009 5:23:15 AM
07/28/2009 05:23:15 AM
07/28/2009 5:23:15
07/28/2009 05:23:15
09/07/28 5:23:15 AM
09/07/28 05:23:15 AM
09/07/28 5:23:15
09/07/28 05:23:15
2009-07-28 5:23:15 AM
2009-07-28 05:23:15 AM
2009-07-28 5:23:15
2009-07-28 05:23:15
28-Jul-09 5:23:15 AM
28-Jul-09 05:23:15 AM
28-Jul-09 5:23:15
28-Jul-09 05:23:15
July 28
July 28
2009-07-28T05:23:15.0160000
2009-07-28T05:23:15.0160000
Tue, 28 Jul 2009 05:23:15 GMT
Tue, 28 Jul 2009 05:23:15 GMT
2009-07-28T05:23:15
5:23 AM
05:23 AM
5:23
05:23
5:23:15 AM
05:23:15 AM
5:23:15
```

```
05:23:15
2009-07-28 05:23:15Z
Tuesday, July 28, 2009 12:23:15 PM
Tuesday, July 28, 2009 12:23:15 PM
Tuesday, July 28, 2009 12:23:15
Tuesday, July 28, 2009 12:23:15
July 28, 2009 12:23:15 PM
July 28, 2009 12:23:15 PM
July 28, 2009 12:23:15
July 28, 2009 12:23:15
Tuesday, 28 July, 2009 12:23:15 PM
Tuesday, 28 July, 2009 12:23:15 PM
Tuesday, 28 July, 2009 12:23:15
Tuesday, 28 July, 2009 12:23:15
28 July, 2009 12:23:15 PM
28 July, 2009 12:23:15 PM
28 July, 2009 12:23:15
28 July, 2009 12:23:15
July, 2009
July, 2009
```

Remarks

The string array returned by the DateTime.GetDateTimeFormats() method is equivalent to combining the string arrays returned by separate calls to the DateTime.GetDateTimeFormats(Char) method with the "d", "D", "f", "F", "g", "G", "m", "o", "r", "s", "t", "T", "u", "U", and "y" standard format strings. For more information about standard format specifiers, see Standard Date and Time Format Strings.

Each element of the return value is formatted using information from the current culture. For more information about culture-specific formatting information for the current culture, see CultureInfo.CurrentCulture.

> **Important**
>
> Because this method uses culture-sensitive data, you should not assume that multiple calls to the method will return identical data. The data returned by this method can change if the current culture changes, the user overrides individual cultural settings, or an update occurs to the system's cultural data.

See also

- Standard Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetDateTimeFormats(Char)

Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier.

C#

```
public string[] GetDateTimeFormats (char format);
```

Parameters

`format` Char

A standard date and time format string.

Returns

String[]

A string array where each element is the representation of the value of this instance formatted with the `format` standard date and time format specifier.

Exceptions

FormatException

`format` is not a valid standard date and time format specifier character.

Examples

The following example demonstrates the GetDateTimeFormats(Char) method. It displays the string representation of a date using the long date format specifier ("D") for the current culture.

C#

```
  DateTime july28 = new DateTime(2009, 7, 28, 5, 23, 15);

  // Get the long date formats using the current culture.
  string [] longJuly28Formats =
              july28.GetDateTimeFormats('D');

  // Display july28 in all long date formats.
  foreach (string format in longJuly28Formats) {
      Console.WriteLine(format);
  }
// The example displays the following output:
//       Tuesday, July 28, 2009
//       July 28, 2009
//       Tuesday, 28 July, 2009
//       28 July, 2009
```

Remarks

The `format` parameter can be any of the standard date and time format specifiers. These include d, D, f, F, g, G, M (or m), O (or o), R (or r), s, t, T, u, U, and Y (or y). For more information, see Standard Date and Time Format Strings.

Each element of the return value is formatted using information from the current culture. For more information about culture-specific formatting information for the current culture, see CultureInfo.CurrentCulture.

> **Important**
>
> Because this method uses culture-sensitive data, you should not assume that multiple calls to the method will return identical data. The data returned by this method can change if the current culture changes, the user overrides individual cultural settings, or an update occurs to the system's cultural data.

See also

- Standard Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetDateTimeFormats(IFormatProvider)

Converts the value of this instance to all the string representations supported by the standard date and time format specifiers and the specified culture-specific formatting information.

C#

```
public string[] GetDateTimeFormats (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information about this instance.

Returns

String[]

A string array where each element is the representation of the value of this instance formatted with one of the standard date and time format specifiers.

Examples

The following example demonstrates the DateTime.GetDateTimeFormats(IFormatProvider) method. It displays the string

representation of a date using all possible standard date and time formats for the fr-FR culture.

```C#
DateTime july28 = new DateTime(2009, 7, 28, 5, 23, 15, 16);

IFormatProvider culture =
    new System.Globalization.CultureInfo("fr-FR", true);
// Get the short date formats using the "fr-FR" culture.
string [] frenchJuly28Formats =
            july28.GetDateTimeFormats(culture);

// Display july28 in various formats using "fr-FR" culture.
foreach (string format in frenchJuly28Formats) {
    Console.WriteLine(format);
}
```

The example displays the following output:

```
28/07/2009
28/07/09
28.07.09
28-07-09
2009-07-28
mardi 28 juillet 2009
28 juil. 09
28 juillet 2009
mardi 28 juillet 2009 05:23
mardi 28 juillet 2009 5:23
mardi 28 juillet 2009 05.23
mardi 28 juillet 2009 05 h 23
28 juil. 09 05:23
28 juil. 09 5:23
28 juil. 09 05.23
28 juil. 09 05 h 23
28 juillet 2009 05:23
28 juillet 2009 5:23
28 juillet 2009 05.23
28 juillet 2009 05 h 23
mardi 28 juillet 2009 05:23:15
mardi 28 juillet 2009 5:23:15
mardi 28 juillet 2009 05.23
mardi 28 juillet 2009 05 h 23
28 juil. 09 05:23:15
28 juil. 09 5:23:15
28 juil. 09 05.23
28 juil. 09 05 h 23
28 juillet 2009 05:23:15
28 juillet 2009 5:23:15
```

```
28 juillet 2009 05.23
28 juillet 2009 05 h 23
28/07/2009 05:23
28/07/2009 5:23
28/07/2009 05.23
28/07/2009 05 h 23
28/07/09 05:23
28/07/09 5:23
28/07/09 05.23
28/07/09 05 h 23
28.07.09 05:23
28.07.09 5:23
28.07.09 05.23
28.07.09 05 h 23
28-07-09 05:23
28-07-09 5:23
28-07-09 05.23
28-07-09 05 h 23
2009-07-28 05:23
2009-07-28 5:23
2009-07-28 05.23
2009-07-28 05 h 23
28/07/2009 05:23:15
28/07/2009 5:23:15
28/07/2009 05.23
28/07/2009 05 h 23
28/07/09 05:23:15
28/07/09 5:23:15
28/07/09 05.23
28/07/09 05 h 23
28.07.09 05:23:15
28.07.09 5:23:15
28.07.09 05.23
28.07.09 05 h 23
28-07-09 05:23:15
28-07-09 5:23:15
28-07-09 05.23
28-07-09 05 h 23
2009-07-28 05:23:15
2009-07-28 5:23:15
2009-07-28 05.23
2009-07-28 05 h 23
28 juillet
28 juillet
2009-07-28T05:23:15.0160000
2009-07-28T05:23:15.0160000
Tue, 28 Jul 2009 05:23:15 GMT
Tue, 28 Jul 2009 05:23:15 GMT
2009-07-28T05:23:15
05:23
5:23
05.23
05 h 23
05:23:15
5:23:15
```

```
05.23
05 h 23
2009-07-28 05:23:15Z
mardi 28 juillet 2009 12:23:15
mardi 28 juillet 2009 12:23:15
mardi 28 juillet 2009 12.23
mardi 28 juillet 2009 12 h 23
28 juil. 09 12:23:15
28 juil. 09 12:23:15
28 juil. 09 12.23
28 juil. 09 12 h 23
28 juillet 2009 12:23:15
28 juillet 2009 12:23:15
28 juillet 2009 12.23
28 juillet 2009 12 h 23
juillet 2009
juillet 2009
```

Remarks

The string array returned by the DateTime.GetDateTimeFormats(IFormatProvider) method is equivalent to combining the string arrays returned by separate calls to the DateTime.GetDateTimeFormats(Char, IFormatProvider) method with the "d", "D", "f", "F", "g", "G", "m", "o", "r", "s", "t", "T", "u", "U", and "y" standard format strings. For more information about standard format specifiers, see Standard Date and Time Format Strings.

Each element of the return value is formatted using culture-specific information supplied by `provider`.

> **Important**
>
> Because this method uses culture-sensitive data, you should not assume that multiple calls to the method will return identical data unless `provider` is a **CultureInfo** object that represents the invariant culture. The data returned by this method can change if the user overrides the individual cultural settings of `provider` or if an update occurs to the system's cultural data for `provider`.

See also

- Standard Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetDateTimeFormats(Char, IFormatProvider)

Converts the value of this instance to all the string representations supported by the specified standard date and time format specifier and culture-specific formatting information.

C#

```
public string[] GetDateTimeFormats (char format, IFormatProvider?
provider);
```

Parameters

`format` Char

A date and time format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about this instance.

Returns

String[]

A string array where each element is the representation of the value of this instance formatted with one of the standard date and time format specifiers.

Exceptions

FormatException

`format` is not a valid standard date and time format specifier character.

Examples

The following example demonstrates the GetDateTimeFormats(Char, IFormatProvider) method. It displays the string representations of a date using the short date format specifier ("d") for the fr-FR culture.

C#

```
  DateTime july28 = new DateTime(2009, 7, 28, 5, 23, 15);

  IFormatProvider culture =
      new System.Globalization.CultureInfo("fr-FR", true);
  // Get the short date formats using the "fr-FR" culture.
  string [] frenchJuly28Formats =
              july28.GetDateTimeFormats('d', culture);

  // Display july28 in short date formats using "fr-FR" culture.
  foreach (string format in frenchJuly28Formats) {
      Console.WriteLine(format);
  }
// The example displays the following output:
//       28/07/2009
//       28/07/09
//       28.07.09
//       28-07-09
//       2009-07-28
```

Remarks

The format parameter can be any of the standard date and time format specifiers. These include d, D, f, F, g, G, M (or m), O (or o), R (or r), s, t, T, u, U, and Y (or y). For more information, see Standard Date and Time Format Strings.

Each element of the return value is formatted using culture-specific information supplied by `provider`.

> **Important**
>
> Because this method uses culture-sensitive data, you should not assume that multiple calls to the method will return identical data unless `provider` is a **CultureInfo** object that represents the invariant culture. The data returned by

this method can change if the user overrides the individual cultural settings of `provider` or if an update occurs to the system's cultural data for `provider`.

See also

- Standard Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTime.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type DateTime.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, DateTime.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTime.IsDaylightSavingTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether this instance of DateTime is within the daylight saving time range for the current time zone.

```
C#
public bool IsDaylightSavingTime ();
```

Returns

Boolean

`true` if the value of the Kind property is Local or Unspecified and the value of this instance of DateTime is within the daylight saving time range for the local time zone; `false` if Kind is Utc.

Remarks

This method determines whether the current DateTime value falls within the daylight saving time range of the local time zone, which is returned by the TimeZoneInfo.Local property. You can determine whether a time zone supports daylight saving time by retrieving the value of its TimeZoneInfo.SupportsDaylightSavingTime property. For time zones that observe daylight saving time, you can determine when the transition to and from daylight saving time occurs by retrieving the System.TimeZoneInfo.AdjustmentRule array returned by the time zone's TimeZoneInfo.GetAdjustmentRules property.

If the current DateTime value represents either an ambiguous or an invalid time in the local time zone, the method returns `false`.

On Windows XP systems, the IsDaylightSavingTime method recognizes only the current adjustment rule when determining whether the current instance is a daylight saving time. As a result, the method may not accurately report whether the current instance is a daylight saving time for periods before the current adjustment rule came into effect.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsDaylightSavingTime(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IsLeapYear(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an indication whether the specified year is a leap year.

C#

```
public static bool IsLeapYear (int year);
```

Parameters

year Int32

A 4-digit year.

Returns

Boolean

`true` if `year` is a leap year; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

Examples

The following example uses the IsLeapYear method to determine which years between 1994 and 2014 are leap years. The example also illustrates the result when the AddYears method is used to add a year to a leap day.

C#

```
using System;

public class IsLeapYear
```

```
{
   public static void Main()
   {
      for (int year = 1994; year <= 2014; year++)
      {
         if (DateTime.IsLeapYear(year))
         {
            Console.WriteLine("{0} is a leap year.", year);
            DateTime leapDay = new DateTime(year, 2, 29);
            DateTime nextYear = leapDay.AddYears(1);
            Console.WriteLine("   One year from {0} is {1}.",
                              leapDay.ToString("d"),
                              nextYear.ToString("d"));
         }
      }
   }
}
// The example produces the following output:
//       1996 is a leap year.
//          One year from 2/29/1996 is 2/28/1997.
//       2000 is a leap year.
//          One year from 2/29/2000 is 2/28/2001.
//       2004 is a leap year.
//          One year from 2/29/2004 is 2/28/2005.
//       2008 is a leap year.
//          One year from 2/29/2008 is 2/28/2009.
//       2012 is a leap year.
//          One year from 2/29/2012 is 2/28/2013.
```

Remarks

`year` is specified as a 4-digit base 10 number; for example, 1996.

`year` is always interpreted as a year in the Gregorian calendar. To determine whether a particular year was a leap year in some other calendar, call that calendar object's `IsLeapYear` method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a date and time to its DateTime equivalent.

Overloads

Expand table

Parse(String)	Converts the string representation of a date and time to its DateTime equivalent by using the conventions of the current culture.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.

Examples

Numerous examples that call the `DateTime.Parse` method are interspersed throughout the Remarks section of this article and in the documentation for the individual `DateTime.Parse` overloads.

Note

Some C# examples in this article run in the **Try.NET** inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

The **local time zone** of the **Try.NET** inline code runner and playground is Coordinated Universal Time, or UTC. This may affect the behavior and the output of examples that illustrate the **DateTime**, **DateTimeOffset**, and **TimeZoneInfo** types and their members.

You can also download a complete set of `DateTime.Parse` examples, which are included in a .NET Core project for C#.

Remarks

In this section:

- Which method do I call?
- The string to parse
- Parsing and cultural conventions
- Parsing and style elements
- The return value and DateTime.Kind

Which method do I call?

Expand table

To	Call
Parse a date and time string by using the conventions of the current culture.	Parse(String) overload
Parse a date and time string by using the conventions of a specific culture.	Parse(String, IFormatProvider) overload (see Parsing and Cultural Conventions)
Parse a date and time string with special style elements (such as white space or no white space).	Parse(String, IFormatProvider, DateTimeStyles) overload

To	Call
Parse a date and time string that must be in a particular format.	DateTime.ParseExact or DateTime.TryParseExact
Parse a date and time string and perform a conversion to UTC or local time.	Parse(String, IFormatProvider, DateTimeStyles) overload
Parse a date and time string without handling exceptions.	DateTime.TryParse method
Restore (round-trip) a date and time value created by a formatting operation.	Pass the "o" or "r" standard format string to the ToString(String) method, and call the Parse(String, IFormatProvider, DateTimeStyles) overload with DateTimeStyles.RoundtripKind
Parse a date and time string in a fixed format across machine (and possibly cultural) boundaries.	DateTime.ParseExact or DateTime.TryParseExact method

The string to parse

The Parse method tries to convert the string representation of a date and time value to its DateTime equivalent. It tries to parse the input string completely without throwing a FormatException exception.

> **ⓘ Important**
>
> If the parsing operation fails because of an unrecognized string format, the **Parse** method throws a **FormatException**, whereas the **TryParse** method returns `false`. Because exception handling can be expensive, you should use **Parse** when the parsing operation is expected to succeed because the input source is trusted. **TryParse** is preferable when parsing failures are likely, particularly because an input source is not trusted, or you have reasonable default values to substitute for strings that do not parse successfully.

The string to be parsed can take any of the following forms:

- A string with a date and a time component.
- A string with a date but no time component. If the time component is absent, the method assumes 12:00 midnight. If the date component has a two-digit year, it is converted to a year based on the Calendar.TwoDigitYearMax of the current

culture's current calendar or the specified culture's current calendar (if you use an overload with a non-null `provider` argument).

- A string with a date component that includes only the month and the year but no day component. The method assumes the first day of the month.
- A string with a date component that includes only the month and the day but no year component. The method assumes the current year.
- A string with a time but no date component. The method assumes the current date unless you call the Parse(String, IFormatProvider, DateTimeStyles) overload and include DateTimeStyles.NoCurrentDateDefault in the `styles` argument, in which case the method assumes a date of January 1, 0001.
- A string with a time component that includes only the hour and an AM/PM designator, with no date component. The method assumes the current date and a time with no minutes and no seconds. You can change this behavior by calling the Parse(String, IFormatProvider, DateTimeStyles) overload and include DateTimeStyles.NoCurrentDateDefault in the `styles` argument, in which case the method assumes a date of January 1, 0001.
- A string that includes time zone information and conforms to ISO 8601. In the following examples, the first string designates Coordinated Universal Time (UTC), and the second designates the time in a time zone that's seven hours earlier than UTC:

  "2008-11-01T19:35:00.0000000Z" "2008-11-01T19:35:00.0000000-07:00"
- A string that includes the GMT designator and conforms to the RFC 1123 time format; for example:

  "Sat, 01 Nov 2008 19:35:00 GMT"
- A string that includes the date and time along with time zone offset information; for example:

  "03/01/2009 05:42:00 -5:00"

The following example parses strings in each of these formats by using the formatting conventions of the current culture, which in this case is the en-US culture:

```C#
using System;

public class Example
```

```
{
   public static void Main()
   {
      (string dateAsString, string description)[]  dateInfo = { ("08/18/2018
07:22:16", "String with a date and time component"),

("08/18/2018", "String with a date component only"),
                                                                  ("8/2018",
"String with a month and year component only"),
                                                                  ("8/18",
"String with a month and day component only"),
                                                                  ("07:22:16",
"String with a time component only"),
                                                                  ("7 PM",
"String with an hour and AM/PM designator only"),
                                                                  ("2018-08-
18T07:22:16.0000000Z", "UTC string that conforms to ISO 8601"),
                                                                  ("2018-08-
18T07:22:16.0000000-07:00", "Non-UTC string that conforms to ISO 8601"),
                                                                  ("Sat, 18
Aug 2018 07:22:16 GMT", "String that conforms to RFC 1123"),
                                                                  ("08/18/2018
07:22:16 -5:00", "String with date, time, and time zone information" ) };

      Console.WriteLine($"Today is {DateTime.Now:d}\n");

      foreach (var item in dateInfo) {
         Console.WriteLine($"{item.description + ":",-52}
'{item.dateAsString}' --> {DateTime.Parse(item.dateAsString)}");
      }
   }
}
// The example displays output like the following:
//   Today is 2/22/2018
//
//   String with a date and time component:               '08/18/2018
07:22:16' --> 8/18/2018 7:22:16 AM
//   String with a date component only:                   '08/18/2018' -->
8/18/2018 12:00:00 AM
//   String with a month and year component only:         '8/2018' -->
8/1/2018 12:00:00 AM
//   String with a month and day component only:          '8/18' -->
8/18/2018 12:00:00 AM
//   String with a time component only:                   '07:22:16' -->
2/22/2018 7:22:16 AM
//   String with an hour and AM/PM designator only:       '7 PM' -->
2/22/2018 7:00:00 PM
//   UTC string that conforms to ISO 8601:                '2018-08-
18T07:22:16.0000000Z' --> 8/18/2018 12:22:16 AM
//   Non-UTC string that conforms to ISO 8601:            '2018-08-
18T07:22:16.0000000-07:00' --> 8/18/2018 7:22:16 AM
//   String that conforms to RFC 1123:                    'Sat, 18 Aug 2018
07:22:16 GMT' --> 8/18/2018 12:22:16 AM
//   String with date, time, and time zone information:   '08/18/2018
07:22:16 -5:00' --> 8/18/2018 5:22:16 AM
```

If the input string represents a leap day in a leap year in the calendar used by the parsing method (see Parsing and cultural conventions), the Parse method parses the string successfully. If the input string represents a leap day in a non-leap year, the method throws a FormatException.

Because the Parse method tries to parse the string representation of a date and time by using the formatting rules of the current or a specified culture, trying to parse a string across different cultures can fail. To parse a specific date and time format across different locales, use one of the overloads of the DateTime.ParseExact method and provide a format specifier.

Parsing and cultural conventions

All overloads of the Parse method are culture-sensitive unless the string to be parsed (which is represented by `s` in the following table) conforms to the ISO 8601 pattern. The parsing operation uses the formatting information in a DateTimeFormatInfo object that is derived as follows:

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

Expand table

If you call	And `provider` is	Formatting information is derived from
Parse(String)	-	The current culture (DateTimeFormatInfo.CurrentInfo property)

If you call	And `provider` is	Formatting information is derived from
Parse(String, IFormatProvider) or Parse(String, IFormatProvider, DateTimeStyles)	a DateTimeFormatInfo object	The specified DateTimeFormatInfo object
Parse(String, IFormatProvider) or Parse(String, IFormatProvider, DateTimeStyles)	`null`	The current culture (DateTimeFormatInfo.CurrentInfo property)
Parse(String, IFormatProvider) or Parse(String, IFormatProvider, DateTimeStyles)	a CultureInfo object	The CultureInfo.DateTimeFormat property
Parse(String, IFormatProvider) or Parse(String, IFormatProvider, DateTimeStyles)	Custom IFormatProvider implementation	The IFormatProvider.GetFormat method

When formatting information is derived from a DateTimeFormatInfo object, the DateTimeFormatInfo.Calendar property defines the calendar used in the parsing operation.

If you parse a date and time string by using a DateTimeFormatInfo object with customized settings that are different from those of a standard culture, use the ParseExact method instead of the Parse method to improve the chances for a successful conversion. A non-standard date and time string can be complicated and difficult to parse. The Parse method tries to parse a string with several implicit parse patterns, all of which might fail. In contrast, the ParseExact method requires you to explicitly designate one or more exact parse patterns that are likely to succeed. For more information, see the "DateTimeFormatInfo and Dynamic Data" section in the DateTimeFormatInfo topic.

> **ⓘ Important**
>
> Note that the formatting conventions for a particular culture are dynamic and can be subject to change. This means that parsing operations that depend on the formatting conventions of the default (current) culture or that specify an

IFormatProvider object that represents a culture other than the invariant culture can unexpectedly fail if any of the following occurs:

- The culture-specific data has changed between major or minor versions of the .NET Framework or as the result of an update to the existing version of the .NET Framework.
- The culture-specific data reflects user preferences, which can vary from machine to machine or session to session.
- The culture-specific data represents a replacement culture that overrides the settings of a standard culture or a custom culture.

To prevent the difficulties in parsing data and time strings that are associated with changes in cultural data, you can parse date and time strings by using the invariant culture, or you can call the **ParseExact** or **TryParseExact** method and specify the exact format of the string to be parsed. If you are serializing and deserializing date and time data, you can either use the formatting conventions of the invariant culture, or you can serialize and deserialize the **DateTime** value in a binary format.

For more information see the "Dynamic culture data" section in the **CultureInfo** topic and the "Persisting DateTime values" section in the **DateTime** topic.

Parsing and style elements

All Parse overloads ignore leading, inner, or trailing white-space characters in the input string (which is represented by `s` in the following table). The date and time can be bracketed with a pair of leading and trailing NUMBER SIGN characters ("#", U+0023), and can be trailed with one or more NULL characters (U+0000).

In addition, the Parse(String, IFormatProvider, DateTimeStyles) overload has a `styles` parameter that consists of one or more members of the DateTimeStyles enumeration. This parameter defines how `s` should be interpreted and how the parse operation should convert `s` to a date and time. The following table describes the effect of each DateTimeStyles member on the parse operation.

Expand table

DateTimeStyles member	Effect on conversion
AdjustToUniversal	Parses `s` and, if necessary, converts it to UTC, as follows:

DateTimeStyles member	Effect on conversion
	- If `s` includes a time zone offset, or if `s` contains no time zone information but `styles` includes the AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc. - If `s` indicates that it represents UTC, or if `s` does not contain time zone information but `styles` includes the AssumeUniversal flag, the method parses the string, performs no time zone conversion on the returned DateTime value, and sets the Kind property to DateTimeKind.Utc. - In all other cases, the flag has no effect.
AllowInnerWhite	This value is ignored. Inner white space is always permitted in the date and time elements of `s`.
AllowLeadingWhite	This value is ignored. Leading white space is always permitted in the date and time elements of `s`.
AllowTrailingWhite	This value is ignored. Trailing white space is always permitted in the date and time elements of `s`.
AllowWhiteSpaces	Specifies that `s` may contain leading, inner, and trailing white spaces. This is the default behavior. It cannot be overridden by supplying a more restrictive DateTimeStyles enumeration value such as None.
AssumeLocal	Specifies that if `s` lacks any time zone information, local time is assumed. Unless the AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local.
AssumeUniversal	Specifies that if `s` lacks any time zone information, UTC is assumed. Unless the AdjustToUniversal flag is present, the method converts the returned DateTime value from UTC to local time and sets its Kind property to DateTimeKind.Local.
None	Although valid, this value is ignored.
RoundtripKind	For strings that contain time zone information, tries to prevent the conversion of a date and time string to a DateTime value that represents a local time with its Kind property set to DateTimeKind.Local. Typically, such a string is created by calling the DateTime.ToString(String) method and by using the "o", "r", or "u" standard format specifier.

The return value and DateTime.Kind

The `DateTime.Parse` overloads return a DateTime value whose Kind property includes time zone information. It can indicate that the time is:

- Coordinated Universal Time (System.DateTimeKind.Utc).

- The time in the local time zone (System.DateTimeKind.Local).
- The time in an unknown time zone (System.DateTimeKind.Unspecified).

Generally, the Parse method returns a DateTime object whose Kind property is DateTimeKind.Unspecified. However, the Parse method may also perform time zone conversion and set the value of the Kind property differently, depending on the values of the `s` and `styles` parameters:

Expand table

If	Time zone conversion	Kind property
`s` contains time zone information.	The date and time is converted to the time in the local time zone.	DateTimeKind.Local
`s` contains time zone information, and `styles` includes the AdjustToUniversal flag.	The date and time is converted to Coordinated Universal Time (UTC).	DateTimeKind.Utc
`s` contains the Z or GMT time zone designator, and `styles` includes the RoundtripKind flag.	The date and time are interpreted as UTC.	DateTimeKind.Utc

The following example converts date strings that contain time zone information to the time in the local time zone:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] dateStrings = {"2008-05-01T07:34:42-5:00",
                              "2008-05-01 7:34:42Z",
                              "Thu, 01 May 2008 07:34:42 GMT"};
      foreach (string dateString in dateStrings)
      {
         DateTime convertedDate = DateTime.Parse(dateString);
         Console.WriteLine($"Converted {dateString} to {convertedDate.Kind}
time {convertedDate}");
      }
   }
}
// These calls to the DateTime.Parse method display the following output:
//  Converted 2008-05-01T07:34:42-5:00 to Local time 5/1/2008 5:34:42 AM
//  Converted 2008-05-01 7:34:42Z to Local time 5/1/2008 12:34:42 AM
```

```
//  Converted Thu, 01 May 2008 07:34:42 GMT to Local time 5/1/2008 12:34:42
AM
```

You can also preserve the value of a date and time's Kind property during a formatting and parsing operation by using the DateTimeStyles.RoundtripKind flag. The following example illustrates how the RoundtripKind flag affects the parsing operation on DateTime values that are converted to strings by using the "o", "r", or "u" format specifier.

C#

```
   string[] formattedDates = { "2008-09-15T09:30:41.7752486-07:00",
                               "2008-09-15T09:30:41.7752486Z",
                               "2008-09-15T09:30:41.7752486",
                               "2008-09-15T09:30:41.7752486-04:00",
                               "Mon, 15 Sep 2008 09:30:41 GMT" };
   foreach (string formattedDate in formattedDates)
   {
      Console.WriteLine(formattedDate);
      DateTime roundtripDate = DateTime.Parse(formattedDate, null,
                                              DateTimeStyles.RoundtripKind);
      Console.WriteLine($"   With RoundtripKind flag: {roundtripDate}
{roundtripDate.Kind} time.");

      DateTime noRoundtripDate = DateTime.Parse(formattedDate, null,
                                                DateTimeStyles.None);
      Console.WriteLine($"   Without RoundtripKind flag: {noRoundtripDate}
{noRoundtripDate.Kind} time.");
   }
// The example displays the following output:
//       2008-09-15T09:30:41.7752486-07:00
//          With RoundtripKind flag: 9/15/2008 9:30:41 AM Local time.
//          Without RoundtripKind flag: 9/15/2008 9:30:41 AM Local time.
//       2008-09-15T09:30:41.7752486Z
//          With RoundtripKind flag: 9/15/2008 9:30:41 AM Utc time.
//          Without RoundtripKind flag: 9/15/2008 2:30:41 AM Local time.
//       2008-09-15T09:30:41.7752486
//          With RoundtripKind flag: 9/15/2008 9:30:41 AM Unspecified time.
//          Without RoundtripKind flag: 9/15/2008 9:30:41 AM Unspecified
time.
//       2008-09-15T09:30:41.7752486-04:00
//          With RoundtripKind flag: 9/15/2008 6:30:41 AM Local time.
//          Without RoundtripKind flag: 9/15/2008 6:30:41 AM Local time.
//       Mon, 15 Sep 2008 09:30:41 GMT
//          With RoundtripKind flag: 9/15/2008 9:30:41 AM Utc time.
//          Without RoundtripKind flag: 9/15/2008 2:30:41 AM Local time.
```

Parse(String)

Converts the string representation of a date and time to its DateTime equivalent by using the conventions of the current culture.

```
C#

public static DateTime Parse (string s);
```

Parameters

s String

A string that contains a date and time to convert. See The string to parse for more information.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date and time.

Examples

The following example parses the string representation of several date and time values by:

- Using the default format provider, which provides the formatting conventions of the current culture of the computer used to produce the example output. The output from this example reflects the formatting conventions of the en-US culture.
- Using the default style value, which is AllowWhiteSpaces.

It handles the FormatException exception that is thrown when the method tries to parse the string representation of a date and time by using some other culture's

formatting conventions. It also shows how to successfully parse a date and time value that does not use the formatting conventions of the current culture.

C#

```
using System;
using System.Globalization;

public class DateTimeParser
{
   public static void Main()
   {
      // Assume the current culture is en-US.
      // The date is February 16, 2008, 12 hours, 15 minutes and 12
seconds.

      // Use standard en-US date and time value
      DateTime dateValue;
      string dateString = "2/16/2008 12:15:12 PM";
      try {
         dateValue = DateTime.Parse(dateString);
         Console.WriteLine("'{0}' converted to {1}.", dateString,
dateValue);
      }
      catch (FormatException) {
         Console.WriteLine("Unable to convert '{0}'.", dateString);
      }

      // Reverse month and day to conform to the fr-FR culture.
      // The date is February 16, 2008, 12 hours, 15 minutes and 12
seconds.
      dateString = "16/02/2008 12:15:12";
      try {
         dateValue = DateTime.Parse(dateString);
         Console.WriteLine("'{0}' converted to {1}.", dateString,
dateValue);
      }
      catch (FormatException) {
         Console.WriteLine("Unable to convert '{0}'.", dateString);
      }

      // Call another overload of Parse to successfully convert string
      // formatted according to conventions of fr-FR culture.
      try {
         dateValue = DateTime.Parse(dateString, new CultureInfo("fr-FR",
false));
         Console.WriteLine("'{0}' converted to {1}.", dateString,
dateValue);
      }
      catch (FormatException) {
         Console.WriteLine("Unable to convert '{0}'.", dateString);
      }

      // Parse string with date but no time component.
```

```
        dateString = "2/16/2008";
        try {
           dateValue = DateTime.Parse(dateString);
           Console.WriteLine("'{0}' converted to {1}.", dateString,
dateValue);
        }
        catch (FormatException) {
           Console.WriteLine("Unable to convert '{0}'.", dateString);
        }
    }
}
// The example displays the following output to the console:
//       '2/16/2008 12:15:12 PM' converted to 2/16/2008 12:15:12 PM.
//       Unable to convert '16/02/2008 12:15:12'.
//       '16/02/2008 12:15:12' converted to 2/16/2008 12:15:12 PM.
//       '2/16/2008' converted to 2/16/2008 12:00:00 AM.
```

Remarks

If `s` contains time zone information, this method returns a DateTime value whose Kind property is DateTimeKind.Local and converts the date and time in `s` to local time. Otherwise, it performs no time zone conversion and returns a DateTime value whose Kind property is DateTimeKind.Unspecified.

This overload attempts to parse `s` by using the formatting conventions of the current culture. The current culture is indicated by the CurrentCulture property. To parse a string using the formatting conventions of a specific culture, call the Parse(String, IFormatProvider) or the Parse(String, IFormatProvider, DateTimeStyles) overloads.

This overload attempts to parse `s` by using DateTimeStyles.AllowWhiteSpaces style.

See also

- TryParse
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static DateTime Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

DateTime

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, IFormatProvider)

Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information.

```
C#

public static DateTime Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a date and time to convert. See The string to parse for more information.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`. See Parsing and cultural conventions

Returns

DateTime

An object that is equivalent to the date and time contained in `s` as specified by `provider`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date and time.

Examples

The following example parses an array of date strings by using the conventions of the en-US, fr-FR, and de-DE cultures. It demonstrates that the string representations of a single date can be interpreted differently across different cultures.

C#

```
using System;
using System.Globalization;

public class ParseDate
{
   public static void Main()
   {
      // Define cultures to be used to parse dates.
      CultureInfo[] cultures = {CultureInfo.CreateSpecificCulture("en-
US"),
                                CultureInfo.CreateSpecificCulture("fr-
FR"),
                                CultureInfo.CreateSpecificCulture("de-
DE")};
      // Define string representations of a date to be parsed.
      string[] dateStrings = {"01/10/2009 7:34 PM",
                              "10.01.2009 19:34",
                              "10-1-2009 19:34" };
      // Parse dates using each culture.
      foreach (CultureInfo culture in cultures)
      {
         DateTime dateValue;
         Console.WriteLine("Attempted conversions using {0} culture.",
                           culture.Name);
         foreach (string dateString in dateStrings)
         {
            try {
               dateValue = DateTime.Parse(dateString, culture);
               Console.WriteLine("   Converted '{0}' to {1}.",
                                 dateString, dateValue.ToString("f",
culture));
            }
            catch (FormatException) {
               Console.WriteLine("   Unable to convert '{0}' for culture
{1}.",
                                 dateString, culture.Name);
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output to the console:
//       Attempted conversions using en-US culture.
//          Converted '01/10/2009 7:34 PM' to Saturday, January 10, 2009
```

```
7:34 PM.
//          Converted '10.01.2009 19:34' to Thursday, October 01, 2009
7:34 PM.
//          Converted '10-1-2009 19:34' to Thursday, October 01, 2009
7:34 PM.
//
//       Attempted conversions using fr-FR culture.
//          Converted '01/10/2009 7:34 PM' to jeudi 1 octobre 2009 19:34.
//          Converted '10.01.2009 19:34' to samedi 10 janvier 2009 19:34.
//          Converted '10-1-2009 19:34' to samedi 10 janvier 2009 19:34.
//
//       Attempted conversions using de-DE culture.
//          Converted '01/10/2009 7:34 PM' to Donnerstag, 1. Oktober 2009
19:34.
//          Converted '10.01.2009 19:34' to Samstag, 10. Januar 2009
19:34.
//          Converted '10-1-2009 19:34' to Samstag, 10. Januar 2009
19:34.
```

Remarks

If `s` contains time zone information, this method returns a DateTime value whose Kind property is DateTimeKind.Local and converts the date and time in `s` to local time. Otherwise, it performs no time zone conversion and returns a DateTime value whose Kind property is DateTimeKind.Unspecified.

This overload attempts to parse `s` by using the DateTimeStyles.AllowWhiteSpaces style.

See also

- TryParse
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts a memory span that contains string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.

C#

```
public static DateTime Parse (ReadOnlySpan<char> s, IFormatProvider?
provider = default, System.Globalization.DateTimeStyles styles =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

The memory span that contains the string to parse. See The string to parse for more information.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`. See Parsing and cultural conventions

`styles` DateTimeStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s` for the parse operation to succeed, and that defines how to interpret the parsed date in relation to the current time zone or the current date. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `provider` and `styles`.

Exceptions

FormatException

`s` does not contain a valid string representation of a date and time.

ArgumentException

`styles` contains an invalid combination of DateTimeStyles values. For example, both AssumeLocal and AssumeUniversal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String, IFormatProvider, DateTimeStyles)

Converts the string representation of a date and time to its DateTime equivalent by using culture-specific format information and a formatting style.

C#

```
public static DateTime Parse (string s, IFormatProvider? provider,
System.Globalization.DateTimeStyles styles);
```

Parameters

s String

A string that contains a date and time to convert. See The string to parse for more information.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. See Parsing and cultural conventions

`styles` DateTimeStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s` for the parse operation to succeed, and that defines how to interpret the parsed date in relation to the current time zone or the current date. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `provider` and `styles`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a date and time.

ArgumentException

`styles` contains an invalid combination of DateTimeStyles values. For example, both AssumeLocal and AssumeUniversal.

Examples

The following example demonstrates the Parse(String, IFormatProvider, DateTimeStyles) method and displays the value of the Kind property of the resulting DateTime values.

C#

```
using System;
using System.Globalization;

public class ParseDateExample
{
   public static void Main()
```

```
    {
       string dateString;
       CultureInfo culture ;
       DateTimeStyles styles;
       DateTime result;

       // Parse a date and time with no styles.
       dateString = "03/01/2009 10:00 AM";
       culture = CultureInfo.CreateSpecificCulture("en-US");
       styles = DateTimeStyles.None;
       try {
          result = DateTime.Parse(dateString, culture, styles);
          Console.WriteLine("{0} converted to {1} {2}.",
                            dateString, result, result.Kind.ToString());
       }
       catch (FormatException) {
          Console.WriteLine("Unable to convert {0} to a date and time.",
                            dateString);
       }

       // Parse the same date and time with the AssumeLocal style.
       styles = DateTimeStyles.AssumeLocal;
       try {
          result = DateTime.Parse(dateString, culture, styles);
          Console.WriteLine("{0} converted to {1} {2}.",
                            dateString, result, result.Kind.ToString());
       }
       catch (FormatException) {
          Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);
       }

       // Parse a date and time that is assumed to be local.
       // This time is five hours behind UTC. The local system's time zone
is
       // eight hours behind UTC.
       dateString = "2009/03/01T10:00:00-5:00";
       styles = DateTimeStyles.AssumeLocal;
       try {
          result = DateTime.Parse(dateString, culture, styles);
          Console.WriteLine("{0} converted to {1} {2}.",
                            dateString, result, result.Kind.ToString());
       }
       catch (FormatException) {
          Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);
       }

       // Attempt to convert a string in improper ISO 8601 format.
       dateString = "03/01/2009T10:00:00-5:00";
       try {
          result = DateTime.Parse(dateString, culture, styles);
          Console.WriteLine("{0} converted to {1} {2}.",
                            dateString, result, result.Kind.ToString());
       }
```

```
      catch (FormatException) {
         Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);
      }

      // Assume a date and time string formatted for the fr-FR culture is
the local
      // time and convert it to UTC.
      dateString = "2008-03-01 10:00";
      culture = CultureInfo.CreateSpecificCulture("fr-FR");
      styles = DateTimeStyles.AdjustToUniversal |
DateTimeStyles.AssumeLocal;
      try {
         result = DateTime.Parse(dateString, culture, styles);
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, result, result.Kind.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);
      }
   }
}
// The example displays the following output to the console:
//       03/01/2009 10:00 AM converted to 3/1/2009 10:00:00 AM
Unspecified.
//       03/01/2009 10:00 AM converted to 3/1/2009 10:00:00 AM Local.
//       2009/03/01T10:00:00-5:00 converted to 3/1/2009 7:00:00 AM Local.
//       Unable to convert 03/01/2009T10:00:00-5:00 to a date and time.
//       2008-03-01 10:00 converted to 3/1/2008 6:00:00 PM Utc.
```

Remarks

This method overload converts the date and time in `s` and sets the Kind property of the returned DateTime value as follows:

Expand table

If	Time zone conversion	Kind property
`s` contains no time zone information.	None.	DateTimeKind.Unspecified
`s` contains time zone information.	To the time in the local time zone	DateTimeKind.Local
`s` contains time zone information, and `styles` includes the DateTimeStyles.AdjustToUniversal flag.	To Coordinated Universal Time (UTC)	DateTimeKind.Utc

If	Time zone conversion	Kind property
`s` contains the Z or GMT time zone designator, and `styles` includes the DateTimeStyles.RoundtripKind.	None.	DateTimeKind.Utc

See also

- TryParse
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- How to: Round-trip Date and Time Values

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly or an exception is thrown.

Overloads

Expand table

ParseExact(String, String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

Remarks

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

ParseExact(String, String, IFormatProvider)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static DateTime ParseExact (string s, string format,
IFormatProvider? provider);
```

Parameters

s String

A string that contains a date and time to convert.

format String

A format specifier that defines the required format of `s`. For more information, see the Remarks section.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `format` and `provider`.

Exceptions

ArgumentNullException

`s` or `format` is `null`.

FormatException

`s` or `format` is an empty string.

-or-

`s` does not contain a date and time that corresponds to the pattern specified in `format`.

-or-

The hour component and the AM/PM designator in `s` do not agree.

Examples

The following example demonstrates the ParseExact method.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string dateString, format;
      DateTime result;
```

```
      CultureInfo provider = CultureInfo.InvariantCulture;

      // Parse date-only value with invariant culture.
      dateString = "06/15/2008";
      format = "d";
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }

      // Parse date-only value without leading zero in month using "d"
format.
      // Should throw a FormatException because standard short date
pattern of
      // invariant culture requires two-digit month.
      dateString = "6/15/2008";
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }

      // Parse date and time with custom specifier.
      dateString = "Sun 15 Jun 2008 8:30 AM -06:00";
      format = "ddd dd MMM yyyy h:mm tt zzz";
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }

      // Parse date and time with offset but without offset's minutes.
      // Should throw a FormatException because "zzz" specifier requires
leading
      // zero in hours.
      dateString = "Sun 15 Jun 2008 8:30 AM -06";
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
```

```
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }

      dateString = "15/06/2008 08:30";
      format = "g";
      provider = new CultureInfo("fr-FR");
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }

      // Parse a date that includes seconds and milliseconds
      // by using the French (France) and invariant cultures.
      dateString = "18/08/2015 06:30:15.006542";
      format = "dd/MM/yyyy HH:mm:ss.ffffff";
      try {
         result = DateTime.ParseExact(dateString, format, provider);
         Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
      }
      catch (FormatException) {
         Console.WriteLine("{0} is not in the correct format.",
dateString);
      }
   }
}
// The example displays the following output:
//       06/15/2008 converts to 6/15/2008 12:00:00 AM.
//       6/15/2008 is not in the correct format.
//       Sun 15 Jun 2008 8:30 AM -06:00 converts to 6/15/2008 7:30:00 AM.
//       Sun 15 Jun 2008 8:30 AM -06 is not in the correct format.
//       15/06/2008 08:30 converts to 6/15/2008 8:30:00 AM.
//       18/08/2015 06:30:15.006542 converts to 8/18/2015 6:30:15 AM.
```

Remarks

The DateTime.ParseExact(String, String, IFormatProvider) method parses the string representation of a date, which must be in the format defined by the `format` parameter. It also requires that the <Date> and <Time> elements of the string representation of a date and time appear in the order specified by `format`, and that `s` have no white space other than that permitted by `format`. If `format` defines a date with no time element and the parse operation succeeds, the resulting DateTime value has a time of midnight (00:00:00). If `format` defines a time with no date

element and the parse operation succeeds, the resulting DateTime value has a date of `DateTime.Now.Date`.

If `s` does not represent a time in a particular time zone and the parse operation succeeds, the Kind property of the returned DateTime value is DateTimeKind.Unspecified. If `s` does represent the time in a particular time zone and `format` allows time zone information to be present (for example, if `format` is equal to the "o", "r", or "u" standard format specifiers, or if it contains the "z", "zz", or "zzz" custom format specifiers), the Kind property of the returned DateTime value is DateTimeKind.Local.

The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of `s`. For details about valid formatting codes, see Standard Date and Time Format Strings or Custom Date and Time Format Strings.

> **Note**
>
> If `format` is a custom format pattern that does not include date or time separators (such as "yyyyMMddHHmm"), use the invariant culture for the `provider` parameter and the widest form of each custom format specifier. For example, if you want to specify hours in the format pattern, specify the wider form, "HH", instead of the narrower form, "H".

The particular date and time symbols and strings (such as names of the days of the week in a particular language) used in `s` are defined by the `provider` parameter, as is the precise format of `s` if `format` is a standard format specifier string. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture used to interpret `s`. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `s`.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom IFormatProvider implementation whose GetFormat method returns either the CultureInfo object or the DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static DateTime ParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider,
System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent a date and time to convert.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a format specifier that defines the required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that provides additional information about `s`, about style elements that may be present in `s`, or about the conversion from `s` to a DateTime value. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `format`, `provider`, and `style`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)

Converts the specified span representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

```
C#

public static DateTime ParseExact (ReadOnlySpan<char> s, string[]
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style = System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent a date and time to convert.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `formats`, `provider`, and `style`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ParseExact(String, String, IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

```
C#

public static DateTime ParseExact (string s, string format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style);
```

Parameters

`s` String

A string containing a date and time to convert.

`format` String

A format specifier that defines the required format of `s`. For more information, see the Remarks section.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that provides additional information about `s`, about style elements that may be present in `s`, or about the conversion from `s` to a DateTime value. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `format`, `provider`, and `style`.

Exceptions

ArgumentNullException

`s` or `format` is `null`.

FormatException

`s` or `format` is an empty string.

-or-

`s` does not contain a date and time that corresponds to the pattern specified in `format`.

-or-

The hour component and the AM/PM designator in `s` do not agree.

ArgumentException

`style` contains an invalid combination of DateTimeStyles values. For example, both AssumeLocal and AssumeUniversal.

Examples

The following example demonstrates the ParseExact(String, String, IFormatProvider) method. Note that the string " 5/01/2009 8:30 AM" cannot be parsed successfully when the `styles` parameter equals DateTimeStyles.None because leading spaces are not allowed by `format`. Additionally, the string "5/01/2009 09:00" cannot be parsed successfully with a `format` of "MM/dd/yyyyhh:mm" because the date string does not precede the month number with a leading zero, as `format` requires.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo enUS = new CultureInfo("en-US");
      string dateString;
```

```
      DateTime dateValue;

      // Parse date with no style flags.
      dateString = " 5/01/2009 8:30 AM";
      try {
         dateValue = DateTime.ParseExact(dateString, "g", enUS,
DateTimeStyles.None);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }
      // Allow a leading space in the date string.
      try {
         dateValue = DateTime.ParseExact(dateString, "g", enUS,
DateTimeStyles.AllowLeadingWhite);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }

      // Use custom formats with M and MM.
      dateString = "5/01/2009 09:00";
      try {
         dateValue = DateTime.ParseExact(dateString, "M/dd/yyyy hh:mm",
enUS, DateTimeStyles.None);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }
      // Allow a leading space in the date string.
      try {
         dateValue = DateTime.ParseExact(dateString, "MM/dd/yyyy hh:mm",
enUS, DateTimeStyles.None);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }

      // Parse a string with time zone information.
```

```
      dateString = "05/01/2009 01:30:42 PM -05:00";
      try {
         dateValue = DateTime.ParseExact(dateString, "MM/dd/yyyy hh:mm:ss
tt zzz", enUS, DateTimeStyles.None);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }
      // Allow a leading space in the date string.
      try {
         dateValue = DateTime.ParseExact(dateString, "MM/dd/yyyy hh:mm:ss
tt zzz", enUS,
                                         DateTimeStyles.AdjustToUniversal);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }

      // Parse a string representing UTC.
      dateString = "2008-06-11T16:11:20.0904778Z";
      try {
         dateValue = DateTime.ParseExact(dateString, "o",
CultureInfo.InvariantCulture,
                                         DateTimeStyles.None);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }
      try {
         dateValue = DateTime.ParseExact(dateString, "o",
CultureInfo.InvariantCulture,
                                         DateTimeStyles.RoundtripKind);
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
      }
   }
}
// The example displays the following output:
```

```
//    ' 5/01/2009 8:30 AM' is not in an acceptable format.
//    Converted ' 5/01/2009 8:30 AM' to 5/1/2009 8:30:00 AM
(Unspecified).
//    Converted '5/01/2009 09:00' to 5/1/2009 9:00:00 AM (Unspecified).
//    '5/01/2009 09:00' is not in an acceptable format.
//    Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 11:30:42 AM
(Local).
//    Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 6:30:42 PM
(Utc).
//    Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 9:11:20 AM
(Local).
//    Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 4:11:20 PM
(Utc).
```

Remarks

The DateTime.ParseExact(String, String, IFormatProvider, DateTimeStyles) method parses the string representation of a date, which must be in a format defined by the `format` parameter. It also requires that the date and time elements in `s` appear in the order specified by `format`. If `s` does not match the pattern of the `format` parameter, with any variations defined by the `style` parameter, the method throws a FormatException. In contrast, the DateTime.Parse(String, IFormatProvider, DateTimeStyles) method parses the string representation of a date in any one of the formats recognized by the format provider's DateTimeFormatInfo object. The DateTime.Parse(String, IFormatProvider, DateTimeStyles) method also allows the date and time elements in `s` to appear in any order.

If the `s` parameter contains only a time and no date, the current date is used unless the `style` parameter includes the DateTimeStyles.NoCurrentDateDefault flag, in which case the default date (`DateTime.Date.MinValue`) is used. If the `s` parameter contains only a date and no time, midnight (00:00:00) is used. The `style` parameter also determines whether the `s` parameter can contain leading, inner, or trailing white space characters.

If `s` contains no time zone information, the Kind property of the returned DateTime object is DateTimeKind.Unspecified. This behavior can be changed by using the DateTimeStyles.AssumeLocal flag, which returns a DateTime value whose Kind property is DateTimeKind.Local, or by using the DateTimeStyles.AssumeUniversal and DateTimeStyles.AdjustToUniversal flags, which returns a DateTime value whose Kind property is DateTimeKind.Utc. If `s` contains time zone information, the time is converted to local time, if necessary, and the Kind property of the returned DateTime object is set to DateTimeKind.Local. This behavior can be changed by using the

DateTimeStyles.RoundtripKind flag to not convert Coordinated Universal Time (UTC) to a local time and to set the Kind property to DateTimeKind.Utc.

The `format` parameter defines the required pattern of the `s` parameter. It can consist of either one or more custom format specifiers from the Custom Date and Time Format Strings table, or a single standard format specifier, which identifies a predefined pattern, from the Standard Date and Time Format Strings table.

If you do not use date or time separators in a custom format pattern, use the invariant culture for the `provider` parameter and the widest form of each custom format specifier. For example, if you want to specify hours in the pattern, specify the wider form, "HH", instead of the narrower form, "H".

> **Note**
>
> Rather than requiring that `s` conform to a single format for the parse operation to succeed, you can call the **DateTime.ParseExact(String, String[], IFormatProvider, DateTimeStyles)** method and specify multiple permitted formats. This makes the parse operation more likely to succeed.

The `styles` parameter includes one or more members of the DateTimeStyles enumeration that determine whether and where white space not defined by `format` can appear in `s` and that control the precise behavior of the parse operation. The following table describes how each member of the DateTimeStyles enumeration affects the operation of the ParseExact(String, String, IFormatProvider, DateTimeStyles) method.

Expand table

DateTimeStyles member	Description
AdjustToUniversal	Parses `s` and, if necessary, converts it to UTC. If `s` includes a time zone offset, or if `s` contains no time zone information but `styles` includes the DateTimeStyles.AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc. If `s` indicates that it represents UTC, or if `s` does not contain time zone information but `styles` includes the DateTimeStyles.AssumeUniversal flag, the method parses the string, performs no time zone conversion on the returned DateTime value, and sets the Kind property to DateTimeKind.Utc. In all other cases, the flag has no effect.

DateTimeStyles member	Description
AllowInnerWhite	Specifies that white space not defined by `format` can appear between any individual date or time element.
AllowLeadingWhite	Specifies that white space not defined by `format` can appear at the beginning of `s`.
AllowTrailingWhite	Specifies that white space not defined by `format` can appear at the end of `s`.
AllowWhiteSpaces	Specifies that `s` may contain leading, inner, and trailing white spaces not defined by `format`.
AssumeLocal	Specifies that if `s` lacks any time zone information, it is assumed to represent a local time. Unless the DateTimeStyles.AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local.
AssumeUniversal	Specifies that if `s` lacks any time zone information, it is assumed to represent UTC. Unless the DateTimeStyles.AdjustToUniversal flag is present, the method converts the returned DateTime value from UTC to local time and sets its Kind property to DateTimeKind.Local.
NoCurrentDateDefault	If `s` contains time without date information, the date of the return value is set to `DateTime.MinValue.Date`.
None	The `s` parameter is parsed using default values. No white space other than that present in `format` is allowed. If `s` lacks a date component, the date of the returned DateTime value is set to 1/1/0001. If `s` contains no time zone information, the Kind property of the returned DateTime object is set to DateTimeKind.Unspecified. If time zone information is present in `s`, the time is converted to local time and the Kind property of the returned DateTime object is set to DateTimeKind.Local.
RoundtripKind	For strings that contain time zone information, tries to prevent the conversion to a DateTime value date and time with its Kind property set to DateTimeKind.Local. This flag primarily prevents the conversion of UTC times to local times.

The particular date and time symbols and strings (such as the names of the days of the week in a particular language) used in `s` are defined by the `provider` parameter, as is the precise format of `s` if `format` is a standard format specifier string. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture used to interpret `s`. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `s`.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom IFormatProvider implementation whose GetFormat method returns either the CultureInfo object or the DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ParseExact(String, String[], IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static DateTime ParseExact (string s, string[] formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style);
```

Parameters

`s` String

A string that contains a date and time to convert.

`formats` String[]

An array of allowable formats of `s`. For more information, see the Remarks section.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

DateTime

An object that is equivalent to the date and time contained in `s`, as specified by `formats`, `provider`, and `style`.

Exceptions

ArgumentNullException

`s` or `formats` is `null`.

FormatException

`s` is an empty string.

-or-

an element of `formats` is an empty string.

-or-

`s` does not contain a date and time that corresponds to any element of `formats`.

-or-

The hour component and the AM/PM designator in `s` do not agree.

ArgumentException

`style` contains an invalid combination of DateTimeStyles values. For example, both AssumeLocal and AssumeUniversal.

Examples

The following example uses the DateTime.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method to ensure that a string in a number of possible formats can be successfully parsed .

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] formats= {"M/d/yyyy h:mm:ss tt", "M/d/yyyy h:mm tt",
                         "MM/dd/yyyy hh:mm:ss", "M/d/yyyy h:mm:ss",
                         "M/d/yyyy hh:mm tt", "M/d/yyyy hh tt",
                         "M/d/yyyy h:mm", "M/d/yyyy h:mm",
                         "MM/dd/yyyy hh:mm", "M/dd/yyyy hh:mm",
                         "MM/d/yyyy HH:mm:ss.ffffff" };
      string[] dateStrings = {"5/1/2009 6:32 PM", "05/01/2009 6:32:05
PM",
```

```
                                "5/1/2009 6:32:00", "05/01/2009 06:32",
                                "05/01/2009 06:32:00 PM", "05/01/2009
06:32:00",
                                "08/28/2015 16:17:39.125", "08/28/2015
16:17:39.125000" };
      DateTime dateValue;

      foreach (string dateString in dateStrings)
      {
         try {
            dateValue = DateTime.ParseExact(dateString, formats,
                                            new CultureInfo("en-US"),
                                            DateTimeStyles.None);
            Console.WriteLine("Converted '{0}' to {1}.", dateString,
dateValue);
         }
         catch (FormatException) {
            Console.WriteLine("Unable to convert '{0}' to a date.",
dateString);
         }
      }
   }
}
// The example displays the following output:
//       Converted '5/1/2009 6:32 PM' to 5/1/2009 6:32:00 PM.
//       Converted '05/01/2009 6:32:05 PM' to 5/1/2009 6:32:05 PM.
//       Converted '5/1/2009 6:32:00' to 5/1/2009 6:32:00 AM.
//       Converted '05/01/2009 06:32' to 5/1/2009 6:32:00 AM.
//       Converted '05/01/2009 06:32:00 PM' to 5/1/2009 6:32:00 PM.
//       Converted '05/01/2009 06:32:00' to 5/1/2009 6:32:00 AM.
//       Unable to convert '08/28/2015 16:17:39.125' to a date.
//       Converted '08/28/2015 16:17:39.125000' to 8/28/2015 4:17:39 PM.
```

Remarks

The DateTime.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method parses the string representation of a date that matches any one of the patterns assigned to the `formats` parameter. If the string `s` does not match any one of these patterns with any of the variations defined by the `styles` parameter, the method throws a FormatException. Aside from comparing `s` to multiple formatting patterns, rather than to a single formatting pattern, this overload behaves identically to the DateTime.ParseExact(String, String, IFormatProvider, DateTimeStyles) method.

The `s` parameter contains the date and time to parse. If the `s` parameter contains only a time and no date, the current date is used unless the `style` parameter includes the DateTimeStyles.NoCurrentDateDefault flag, in which case the default date (`DateTime.Date.MinValue`) is used. If the `s` parameter contains only a date and no time, midnight (00:00:00) is used. The `style` parameter also determines whether

the `s` parameter can contain leading, inner, or trailing white space characters other than those permitted by one of the format strings in `formats`.

If `s` contains no time zone information, the Kind property of the returned DateTime object is DateTimeKind.Unspecified. This behavior can be changed by using the DateTimeStyles.AssumeLocal flag, which returns a DateTime value whose Kind property is DateTimeKind.Local, or by using the DateTimeStyles.AssumeUniversal and DateTimeStyles.AdjustToUniversal flags, which returns a DateTime value whose Kind property is DateTimeKind.Utc. If `s` contains time zone information, the time is converted to local time, if necessary, and the Kind property of the returned DateTime object is set to DateTimeKind.Local. This behavior can be changed by using the DateTimeStyles.RoundtripKind flag to not convert Coordinated Universal Time (UTC) to a local time and set the Kind property to DateTimeKind.Utc.

The `formats` parameter contains an array of patterns, one of which `s` must match exactly if the parse operation is to succeed. The patterns in the `formats` parameter consists of one or more custom format specifiers from the Custom Date and Time Format Strings table, or a single standard format specifier, which identifies a predefined pattern, from the Standard Date and Time Format Strings table.

If you do not use date or time separators in a custom format pattern, use the invariant culture for the `provider` parameter and the widest form of each custom format specifier. For example, if you want to specify hours in the pattern, specify the wider form, "HH", instead of the narrower form, "H".

The `styles` parameter includes one or more members of the DateTimeStyles enumeration that determine whether and where white space not defined by `format` can appear in `s` and that control the precise behavior of the parse operation. The following table describes how each member of the DateTimeStyles enumeration affects the operation of the ParseExact(String, String, IFormatProvider, DateTimeStyles) method.

Expand table

DateTimeStyles member	Description
AdjustToUniversal	Parses `s` and, if necessary, converts it to UTC. If `s` includes a time zone offset, or if `s` contains no time zone information but `styles` includes the DateTimeStyles.AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc. If `s` indicates that it represents UTC, or if `s` does not contain time zone

DateTimeStyles member	Description
	information but `styles` includes the DateTimeStyles.AssumeUniversal flag, the method parses the string, performs no time zone conversion on the returned DateTime value, and sets the Kind property to DateTimeKind.Utc. In all other cases, the flag has no effect.
AllowInnerWhite	Specifies that white space not defined by `format` can appear between any individual date or time element.
AllowLeadingWhite	Specifies that white space not defined by `format` can appear at the beginning of `s`.
AllowTrailingWhite	Specifies that white space not defined by `format` can appear at the end of `s`.
AllowWhiteSpaces	Specifies that `s` may contain leading, inner, and trailing white spaces not defined by `format`.
AssumeLocal	Specifies that if `s` lacks any time zone information, it is assumed to represent a local time. Unless the DateTimeStyles.AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local.
AssumeUniversal	Specifies that if `s` lacks any time zone information, it is assumed to represent UTC. Unless the DateTimeStyles.AdjustToUniversal flag is present, the method converts the returned DateTime value from UTC to local time and sets its Kind property to DateTimeKind.Local.
NoCurrentDateDefault	If `s` contains time without date information, the date of the return value is set to `DateTime.MinValue.Date`.
None	The `s` parameter is parsed using default values. No white space other than that present in `format` is allowed. If `s` lacks a date component, the date of the returned DateTime value is set to 1/1/0001. If `s` contains no time zone information, the Kind property of the returned DateTime object is set to DateTimeKind.Unspecified. If time zone information is present in `s`, the time is converted to local time and the Kind property of the returned DateTime object is set to DateTimeKind.Local.
RoundtripKind	For strings that contain time zone information, tries to prevent the conversion to a date and time with its Kind property set to DateTimeKind.Local. This flag primarily prevents the conversion of UTC times to local times.

The particular date and time symbols and strings (such as the names of the days of the week in a particular language) used in `s` are defined by the `provider` parameter,

as is the precise format of `s` if `format` is a standard format specifier string. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture used to interpret `s`. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `s`.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom IFormatProvider implementation whose GetFormat method returns either the CultureInfo object or the DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.SpecifyKind(DateTime, DateTimeKind) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new DateTime object that has the same number of ticks as the specified DateTime, but is designated as either local time, Coordinated Universal Time (UTC), or neither, as indicated by the specified DateTimeKind value.

C#

```
public static DateTime SpecifyKind (DateTime value, DateTimeKind kind);
```

Parameters

`value` DateTime

A date and time.

`kind` DateTimeKind

One of the enumeration values that indicates whether the new object represents local time, UTC, or neither.

Returns

DateTime

A new object that has the same number of ticks as the object represented by the `value` parameter and the DateTimeKind value specified by the `kind` parameter.

Examples

The following example uses the SpecifyKind method to demonstrate how the Kind property influences the ToLocalTime and ToUniversalTime conversion methods.

C#

```
// This code example demonstrates the DateTime Kind, Now, and
// UtcNow properties, and the SpecifyKind(), ToLocalTime(),
// and ToUniversalTime() methods.

using System;

class Sample
{
    public static void Main()
    {
        // Get the date and time for the current moment, adjusted
        // to the local time zone.

        DateTime saveNow = DateTime.Now;

        // Get the date and time for the current moment expressed
        // as coordinated universal time (UTC).

        DateTime saveUtcNow = DateTime.UtcNow;
        DateTime myDt;

        // Display the value and Kind property of the current moment
        // expressed as UTC and local time.

        DisplayNow("UtcNow: ..........", saveUtcNow);
        DisplayNow("Now: .............", saveNow);
        Console.WriteLine();

        // Change the Kind property of the current moment to
        // DateTimeKind.Utc and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Utc);
        Display("Utc: .............", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Local and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Local);
        Display("Local: ...........", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Unspecified and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Unspecified);
        Display("Unspecified: .....", myDt);
    }

    // Display the value and Kind property of a DateTime structure, the
    // DateTime structure converted to local time, and the DateTime
    // structure converted to universal time.

    public static string datePatt = @"M/d/yyyy hh:mm:ss tt";
    public static void Display(string title, DateTime inputDt)
    {
```

```
        DateTime dispDt = inputDt;
        string dtString;

        // Display the original DateTime.

        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, dispDt.Kind);

        // Convert inputDt to local time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was universal time.

        dispDt = inputDt.ToLocalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToLocalTime:     {0}, Kind = {1}",
                          dtString, dispDt.Kind);

        // Convert inputDt to universal time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was local time.

        dispDt = inputDt.ToUniversalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToUniversalTime: {0}, Kind = {1}",
                          dtString, dispDt.Kind);
        Console.WriteLine();
    }

    // Display the value and Kind property for DateTime.Now and
DateTime.UtcNow.

    public static void DisplayNow(string title, DateTime inputDt)
    {
        string dtString = inputDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, inputDt.Kind);
    }
}

/*
This code example produces the following results:

UtcNow: .......... 5/6/2005 09:34:42 PM, Kind = Utc
Now: ............. 5/6/2005 02:34:42 PM, Kind = Local

Utc: ............. 5/6/2005 02:34:42 PM, Kind = Utc
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
```

```
  ToUniversalTime: 5/6/2005 02:34:42 PM, Kind = Utc

Local: ........... 5/6/2005 02:34:42 PM, Kind = Local
  ToLocalTime:     5/6/2005 02:34:42 PM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

Unspecified: ..... 5/6/2005 02:34:42 PM, Kind = Unspecified
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

*/
```

Remarks

A DateTime object consists of a Kind field that indicates whether the time value is based on local time, Coordinated Universal Time (UTC), or neither, and a Ticks field that contains a time value measured in 100-nanosecond ticks. The SpecifyKind method creates a new DateTime object using the specified `kind` parameter and the original time value.

> **Important**
>
> The returned **DateTime** value does not represent the same instant in time as the `value` parameter, and **SpecifyKind** is not a time zone conversion method. Instead, it leaves the time specified by the `value` parameter unchanged, and sets the **Kind** property to `kind`. For information about time zone conversions, see **Converting Times Between Time Zones**.

The SpecifyKind method is useful in interoperability scenarios where you receive a DateTime object with an unspecified Kind field, but you can determine by independent means that the Ticks field represents local time or UTC.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- COM Interoperability in .NET Framework Applications (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Subtract Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the value that results from subtracting the specified time or duration from the value of this instance.

Overloads

Expand table

Subtract(TimeSpan)	Returns a new DateTime that subtracts the specified duration from the value of this instance.
Subtract(DateTime)	Returns a new TimeSpan that subtracts the specified date and time from the value of this instance.

Subtract(TimeSpan)

Returns a new DateTime that subtracts the specified duration from the value of this instance.

C#

```
public DateTime Subtract (TimeSpan value);
```

Parameters

`value` TimeSpan

The time interval to subtract.

Returns

DateTime

An object that is equal to the date and time represented by this instance minus the time interval represented by `value`.

Exceptions

ArgumentOutOfRangeException

The result is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates the Subtract method and the subtraction operator.

C#

```
System.DateTime date1 = new System.DateTime(1996, 6, 3, 22, 15, 0);
System.DateTime date2 = new System.DateTime(1996, 12, 6, 13, 2, 0);
System.DateTime date3 = new System.DateTime(1996, 10, 12, 8, 42, 0);

// diff1 gets 185 days, 14 hours, and 47 minutes.
System.TimeSpan diff1 = date2.Subtract(date1);

// date4 gets 4/9/1996 5:55:00 PM.
System.DateTime date4 = date3.Subtract(diff1);

// diff2 gets 55 days 4 hours and 20 minutes.
System.TimeSpan diff2 = date2 - date3;

// date5 gets 4/9/1996 5:55:00 PM.
System.DateTime date5 = date1 - diff2;
```

Remarks

The Subtract(TimeSpan) method returns the date that is a specified time interval difference from the current instance. To determine the time interval between two dates, call the Subtract(DateTime) method. To subtract a particular time interval from the current instance, call the method that adds that time interval to the current date, and supply a negative value as the method argument. For example, to subtract two months from the current date, call the AddMonths(Int32) method with a value of -2.

This method does not change the value of this DateTime. Instead, it returns a new DateTime whose value is the result of this operation.

Ordinarily, the DateTime.Subtract(TimeSpan) method subtracts a TimeSpan object that represents a positive time span and returns a DateTime value that is earlier than

the date and time of the current instance. However, if the TimeSpan object represents a negative time span, the DateTime.Subtract(TimeSpan) method returns a DateTime value that is later than the date and time of the current instance.

The DateTime.Subtract(TimeSpan) method allows you to subtract a time interval that consists of more than one unit of time (such as a given number of hours and a given number of minutes). To subtract a single unit of time (such as years, months, or days) from the DateTime instance, you can pass a negative numeric value as a parameter to any of the following methods:

- AddYears, to subtract a specific number of years from the current date and time instance.
- AddMonths, to subtract a specific number of months from the current date and time instance.
- AddDays, to subtract a specific number of days from the current date and time instance.
- AddHours, to subtract a specific number of hours from the current date and time instance.
- AddMinutes, to subtract a specific number of minutes from the current date and time instance.
- AddSeconds, to subtract a specific number of seconds from the current date and time instance.
- AddMilliseconds, to subtract a specific number of milliseconds from the current date and time instance.
- AddTicks, to subtract a specific number of ticks from the current date and time instance.

See also

- TimeSpan
- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Subtract(DateTime)

Returns a new TimeSpan that subtracts the specified date and time from the value of this instance.

```
C#

public TimeSpan Subtract (DateTime value);
```

Parameters

`value` DateTime

The date and time value to subtract.

Returns

TimeSpan

A time interval that is equal to the date and time represented by this instance minus the date and time represented by `value`.

Exceptions

ArgumentOutOfRangeException

The result is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates the Subtract method and the subtraction operator.

C#

```
System.DateTime date1 = new System.DateTime(1996, 6, 3, 22, 15, 0);
System.DateTime date2 = new System.DateTime(1996, 12, 6, 13, 2, 0);
System.DateTime date3 = new System.DateTime(1996, 10, 12, 8, 42, 0);

// diff1 gets 185 days, 14 hours, and 47 minutes.
System.TimeSpan diff1 = date2.Subtract(date1);

// date4 gets 4/9/1996 5:55:00 PM.
System.DateTime date4 = date3.Subtract(diff1);

// diff2 gets 55 days 4 hours and 20 minutes.
System.TimeSpan diff2 = date2 - date3;

// date5 gets 4/9/1996 5:55:00 PM.
System.DateTime date5 = date1 - diff2;
```

Remarks

The Subtract(DateTime) method determines the difference between two dates. To subtract a time interval from the current instance, call the Subtract(TimeSpan) method. To subtract a particular time interval from the current instance, call the method that adds that time interval to the current date, and supply a negative value as the method argument. For example, to subtract two months from the current date, call the AddMonths(Int32) method with a value of -2.

If the date and time of the current instance is earlier than `value`, the method returns a TimeSpan object that represents a negative time span. That is, the value of all of its non-zero properties (such as Days or Ticks) is negative.

The Subtract(DateTime) method does not consider the value of the Kind property of the two DateTime values when performing the subtraction. Before subtracting DateTime objects, ensure that the objects represent times in the same time zone. Otherwise, the result will include the difference between time zones.

> **Note**
>
> The **DateTimeOffset.Subtract(DateTimeOffset)** method does consider the difference between time zones when performing the subtraction.

See also

- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToBinary Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serializes the current DateTime object to a 64-bit binary value that subsequently can be used to recreate the DateTime object.

```
C#
public long ToBinary ();
```

Returns

Int64

A 64-bit signed integer that encodes the Kind and Ticks properties.

Remarks

For more information about this API, see Supplemental API remarks for DateTime.ToBinary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FromBinary(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToFileTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to a Windows file time.

C#

```
public long ToFileTime ();
```

Returns

Int64

The value of the current DateTime object expressed as a Windows file time.

Exceptions

ArgumentOutOfRangeException

The resulting file time would represent a date and time before 12:00 midnight January 1, 1601 C.E. UTC.

Examples

The following example demonstrates the ToFileTime method.

C#

```
static void Main(string[] args)
{
    System.Console.WriteLine("Enter the file path:");
    string filePath = System.Console.ReadLine();

    if (System.IO.File.Exists(filePath)) {
        System.DateTime fileCreationDateTime =
            System.IO.File.GetCreationTime(filePath);

        long fileCreationFileTime = fileCreationDateTime.ToFileTime();
```

```
        System.Console.WriteLine("{0} in file time is {1}.",
                                 fileCreationDateTime,
                                 fileCreationFileTime);
    }
    else {
        System.Console.WriteLine("{0} is an invalid file", filePath);
    }
}
```

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

The ToFileTime method uses the Kind property to determine whether the current DateTime object is a local time, a UTC time, or an unspecified kind of time which is treated as a local time.

Notes to Callers

Ordinarily, the FromFileTime(Int64) method restores a DateTime value that was saved by the ToFileTime() method. However, the two values may differ under the following conditions:

- If the serialization and deserialization of the DateTime value occur in different time zones. For example, if a DateTime value with a time of 12:30 P.M. in the U.S. Eastern Time zone is serialized, and then deserialized in the U.S. Pacific Time zone, the original value of 12:30 P.M. is adjusted to 9:30 A.M. to reflect the difference between the two time zones.

- If the DateTime value that is serialized represents an invalid time in the local time zone. In this case, the ToFileTime() method adjusts the restored DateTime value so that it represents a valid time in the local time zone.

  For example, the transition from standard time to daylight saving time occurs in the U.S. Pacific Time zone on March 14, 2010, at 2:00 A.M., when the time advances by one hour, to 3:00 A.M. This hour interval is an invalid time, that is, a time interval that does not exist in this time zone. The following example shows that when a time that falls within this range is converted to a long integer value by the ToFileTime() method and is then restored by the FromFileTime(Int64) method, the original value is adjusted to become a valid time. You can determine whether a

particular date and time value may be subject to modification by passing it to the IsInvalidTime(DateTime) method, as the example illustrates.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime date1 = new DateTime(2010, 3, 14, 2, 30, 00);
      Console.WriteLine("Invalid Time: {0}",
                        TimeZoneInfo.Local.IsInvalidTime(date1));
      long ft = date1.ToFileTime();
      DateTime date2 = DateTime.FromFileTime(ft);
      Console.WriteLine("{0} -> {1}", date1, date2);
   }
}
// The example displays the following output:
//       Invalid Time: True
//       3/14/2010 2:30:00 AM -> 3/14/2010 3:30:00 AM
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToFileTimeUtc()
- Int64
- ToUniversalTime()
- GetUtcOffset(DateTime)

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTime.ToFileTimeUtc Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to a Windows file time.

```
C#
public long ToFileTimeUtc ();
```

Returns

Int64

The value of the current DateTime object expressed as a Windows file time.

Exceptions

ArgumentOutOfRangeException

The resulting file time would represent a date and time before 12:00 midnight January 1, 1601 C.E. UTC.

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

The ToFileTimeUtc method uses the Kind property to determine whether the current DateTime object is a local time, a UTC time, or an unspecified kind of time which is treated as a UTC time. If it is a local time, it converts the time to UTC before performing the conversion to a Windows file time.

Notes to Callers

The ToFileTimeUtc() method is sometimes used to convert a local time to UTC, and subsequently to restore it by calling the FromFileTimeUtc(Int64) method followed by the ToLocalTime() method. However, if the original time represents an invalid time in the local time zone, the two local time values will not be equal. For additional information, see the ToLocalTime() method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToFileTime()
- Int64
- ToUniversalTime()
- GetUtcOffset(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToLocalTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to local time.

C#

```
public DateTime ToLocalTime ();
```

Returns

DateTime

An object whose Kind property is Local, and whose value is the local time equivalent to the value of the current DateTime object, or DateTime.MaxValue if the converted value is too large to be represented by a DateTime object, or DateTime.MinValue if the converted value is too small to be represented as a DateTime object.

Examples

The following example demonstrates the ToLocalTime method. Note that the exact output depends on the current culture and the local time zone of the system on which it is run.

C#

```
using System;

class Example
{
    static void Main()
    {
        DateTime localDateTime, univDateTime;

        Console.WriteLine("Enter a date and time.");
        string strDateTime = Console.ReadLine();

        try {
```

```
            localDateTime = DateTime.Parse(strDateTime);
            univDateTime = localDateTime.ToUniversalTime();

            Console.WriteLine("{0} local time is {1} universal time.",
                              localDateTime,
                                  univDateTime);
        }
        catch (FormatException) {
            Console.WriteLine("Invalid format.");
            return;
        }

        Console.WriteLine("Enter a date and time in universal time.");
        strDateTime = Console.ReadLine();

        try {
            univDateTime = DateTime.Parse(strDateTime);
            localDateTime = univDateTime.ToLocalTime();

            Console.WriteLine("{0} universal time is {1} local time.",
                                univDateTime,
                                localDateTime);
        }
        catch (FormatException) {
            Console.WriteLine("Invalid format.");
            return;
        }
    }
}
// The example displays output like the following when run on a
// computer whose culture is en-US in the Pacific Standard Time zone:
//     Enter a date and time.
//     12/10/2015 6:18 AM
//     12/10/2015 6:18:00 AM local time is 12/10/2015 2:18:00 PM universal
time.
//     Enter a date and time in universal time.
//     12/20/2015 6:42:00
//     12/20/2015 6:42:00 AM universal time is 12/19/2015 10:42:00 PM local
time.
```

The following example uses the SpecifyKind method to demonstrate how the Kind property influences the ToLocalTime and ToUniversalTime conversion methods.

C#

```
// This code example demonstrates the DateTime Kind, Now, and
// UtcNow properties, and the SpecifyKind(), ToLocalTime(),
// and ToUniversalTime() methods.

using System;

class Sample
{
```

```
    public static void Main()
    {
        // Get the date and time for the current moment, adjusted
        // to the local time zone.

        DateTime saveNow = DateTime.Now;

        // Get the date and time for the current moment expressed
        // as coordinated universal time (UTC).

        DateTime saveUtcNow = DateTime.UtcNow;
        DateTime myDt;

        // Display the value and Kind property of the current moment
        // expressed as UTC and local time.

        DisplayNow("UtcNow: ..........", saveUtcNow);
        DisplayNow("Now: .............", saveNow);
        Console.WriteLine();

        // Change the Kind property of the current moment to
        // DateTimeKind.Utc and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Utc);
        Display("Utc: .............", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Local and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Local);
        Display("Local: ...........", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Unspecified and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Unspecified);
        Display("Unspecified: .....", myDt);
    }

    // Display the value and Kind property of a DateTime structure, the
    // DateTime structure converted to local time, and the DateTime
    // structure converted to universal time.

    public static string datePatt = @"M/d/yyyy hh:mm:ss tt";
    public static void Display(string title, DateTime inputDt)
    {
        DateTime dispDt = inputDt;
        string dtString;

        // Display the original DateTime.

        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, dispDt.Kind);
```

```
        // Convert inputDt to local time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was universal time.

        dispDt = inputDt.ToLocalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToLocalTime:     {0}, Kind = {1}",
                          dtString, dispDt.Kind);

        // Convert inputDt to universal time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was local time.

        dispDt = inputDt.ToUniversalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToUniversalTime: {0}, Kind = {1}",
                          dtString, dispDt.Kind);
        Console.WriteLine();
    }

    // Display the value and Kind property for DateTime.Now and
DateTime.UtcNow.

    public static void DisplayNow(string title, DateTime inputDt)
    {
        string dtString = inputDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, inputDt.Kind);
    }
}

/*
This code example produces the following results:

UtcNow: .......... 5/6/2005 09:34:42 PM, Kind = Utc
Now: ............. 5/6/2005 02:34:42 PM, Kind = Local

Utc: ............. 5/6/2005 02:34:42 PM, Kind = Utc
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 02:34:42 PM, Kind = Utc

Local: ........... 5/6/2005 02:34:42 PM, Kind = Local
  ToLocalTime:     5/6/2005 02:34:42 PM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

Unspecified: ..... 5/6/2005 02:34:42 PM, Kind = Unspecified
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc
```

```
*/
```

Remarks

The local time is equal to the Coordinated Universal Time (UTC) time plus the UTC offset. For more information about the UTC offset, see TimeZoneInfo.GetUtcOffset. The conversion also takes into account the daylight saving time rule that applies to the time represented by the current DateTime object.

> **Important**
>
> On Windows XP systems, the **ToLocalTime** method recognizes only the current adjustment rule when converting from UTC to local time. As a result, conversions for periods before the current adjustment rule came into effect may not accurately reflect the difference between UTC and local time.

Starting with the .NET Framework version 2.0, the value returned by the ToLocalTime method is determined by the Kind property of the current DateTime object. The following table describes the possible results.

Expand table

Kind	Results
Utc	This instance of DateTime is converted to local time.
Local	No conversion is performed.
Unspecified	This instance of DateTime is assumed to be a UTC time, and the conversion is performed as if Kind were Utc.

> **Note**
>
> The **ToLocalTime** method converts a **DateTime** value from UTC to local time. To convert the time in any designated time zone to local time, use the **TimeZoneInfo.ConvertTime** method.

The value returned by the conversion is a DateTime whose Kind property always returns Local. Consequently, a valid result is returned even if ToLocalTime is applied repeatedly to the same DateTime.

Notes to Callers

You can use the ToLocalTime() method to restore a local date and time value that was converted to UTC by the ToUniversalTime() or FromFileTimeUtc(Int64) method. However, if the original time represents an invalid time in the local time zone, it will not match the restored value. When the ToLocalTime() method converts a time from UTC to the local time zone, it also adjusts the time so that is valid in the local time zone.

For example, the transition from standard time to daylight saving time occurs in the U.S. Pacific Time zone on March 14, 2010, at 2:00 A.M., when the time advances by one hour, to 3:00 A.M. This hour interval is an invalid time, that is, a time interval that does not exist in this time zone. The following example shows that when a time that falls within this range is converted to UTC by the ToUniversalTime() method and is then restored by the ToLocalTime() method, the original value is adjusted to become a valid time. You can determine whether a particular date and time value may be subject to modification by passing it to the IsInvalidTime(DateTime) method, as the example illustrates.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime date1 = new DateTime(2010, 3, 14, 2, 30, 0,
DateTimeKind.Local);
      Console.WriteLine("Invalid time: {0}",
                        TimeZoneInfo.Local.IsInvalidTime(date1));
      DateTime utcDate1 = date1.ToUniversalTime();
      DateTime date2 = utcDate1.ToLocalTime();
      Console.WriteLine("{0} --> {1}", date1, date2);
   }
}
// The example displays the following output:
//       Invalid time: True
//       3/14/2010 2:30:00 AM --> 3/14/2010 3:30:00 AM
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TimeZoneInfo
- GetUtcOffset(DateTime)
- GetAdjustmentRules()
- ToUniversalTime()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToLongDateString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to its equivalent long date string representation.

C#

```
public string ToLongDateString ();
```

Returns

String

A string that contains the long date string representation of the current DateTime object.

Examples

The following example demonstrates the ToLongDateString method.

C#

```
using System;
using System.Globalization;

class Sample
{
    public static void Main()
    {
        // Initialize a DateTime object.
        Console.WriteLine("Initialize the DateTime object to May 16, 2001
3:02:15 AM.\n");
        DateTime dateAndTime = new System.DateTime(2001, 5, 16, 3, 2, 15);

        // Display the name of the current culture.
        Console.WriteLine($"Current culture: \"
{CultureInfo.CurrentCulture.Name}\"\n");
        var dtfi = CultureInfo.CurrentCulture.DateTimeFormat;
```

```
        // Display the long date pattern and string.
        Console.WriteLine($"Long date pattern: \"{dtfi.LongDatePattern}\"");
        Console.WriteLine($"Long date string:  \"
{dateAndTime.ToLongDateString()}\"\n");

        // Display the long time pattern and string.
        Console.WriteLine($"Long time pattern: \"{dtfi.LongTimePattern}\"");
        Console.WriteLine($"Long time string:  \"
{dateAndTime.ToLongTimeString()}\"\n");

        // Display the short date pattern and string.
        Console.WriteLine($"Short date pattern: \"
{dtfi.ShortDatePattern}\"");
        Console.WriteLine($"Short date string:  \"
{dateAndTime.ToShortDateString()}\"\n");

        // Display the short time pattern and string.
        Console.WriteLine($"Short time pattern: \"
{dtfi.ShortTimePattern}\"");
        Console.WriteLine($"Short time string:  \"
{dateAndTime.ToShortTimeString()}\"\n");
    }
}
// The example displays output similar to the following:
//        Current culture: "en-US"
//
//        Long date pattern: "dddd, MMMM d, yyyy"
//        Long date string:  "Wednesday, May 16, 2001"
//
//        Long time pattern: "h:mm:ss tt"
//        Long time string:  "3:02:15 AM"
//
//        Short date pattern: "M/d/yyyy"
//        Short date string:  "5/16/2001"
//
//        Short time pattern: "h:mm tt"
//        Short time string:  "3:02 AM"
```

Remarks

The value of the current DateTime object is formatted using the pattern defined by the LongDatePattern property associated with the current culture. The return value is identical to the value returned by specifying the "D" standard DateTime format string with the ToString(String) method.

> ⓘ **Note**

The string returned by the **ToLongDateString** method is culture-sensitive. It reflects the pattern defined by the current culture's **DateTimeFormatInfo.LongDatePattern** property. For example, for the en-US culture, the standard long date pattern is commonly "dddd, MMMM dd, yyyy"; for the de-DE culture, it is "dddd, d. MMMM yyyy"; for the ja-JP culture, it is "yyyy'年'M'月'd'日'". Note that its value can vary depending on the .NET implementation and its version, the operating system and its version, and user customization.

For more information about the current thread culture, see the CultureInfo.CurrentCulture and Thread.CurrentCulture properties. For more information about format characters, format patterns, and the output they produce, see Standard date and time format strings and Custom date and time format strings. For more information about changing the format pattern associated with a format character, see the DateTimeFormatInfo class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- CultureInfo
- DateTimeFormat
- DateTimeFormatInfo
- ToString()
- Formatting Types in .NET
- Custom Date and Time Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTime.ToLongTimeString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to its equivalent long time string representation.

C#

```
public string ToLongTimeString ();
```

Returns

String

A string that contains the long time string representation of the current DateTime object.

Examples

The following example demonstrates the ToLongTimeString method.

C#

```
using System;
using System.Threading;
using System.Globalization;

public class Sample
{
   public static void Main()
   {
      // Create an array of culture names.
      String[] names = { "en-US", "en-GB", "fr-FR", "de-DE" };
      // Initialize a DateTime object.
      DateTime dateValue = new System.DateTime(2013, 5, 28, 10, 30, 15);

      // Iterate the array of culture names.
      foreach (var name in names) {
         // Change the culture of the current thread.
```

```
        Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture(name);
        // Display the name of the current culture and the date.
        Console.WriteLine("Current culture: {0}",
CultureInfo.CurrentCulture.Name);
        Console.WriteLine("Date: {0:G}", dateValue);

        // Display the long time pattern and the long time.
        Console.WriteLine("Long time pattern: '{0}'",
                          DateTimeFormatInfo.CurrentInfo.LongTimePattern);
        Console.WriteLine("Long time with format string:     {0:T}",
dateValue);
        Console.WriteLine("Long time with ToLongTimeString:  {0}\n",
                          dateValue.ToLongTimeString());
      }
   }
}
// The example displays the following output:
//       Current culture: en-US
//       Date: 5/28/2013 10:30:15 AM
//       Long time pattern: 'h:mm:ss tt'
//       Long time with format string:     10:30:15 AM
//       Long time with ToLongTimeString:  10:30:15 AM
//
//       Current culture: en-GB
//       Date: 28/05/2013 10:30:15
//       Long time pattern: 'HH:mm:ss'
//       Long time with format string:     10:30:15
//       Long time with ToLongTimeString:  10:30:15
//
//       Current culture: fr-FR
//       Date: 28/05/2013 10:30:15
//       Long time pattern: 'HH:mm:ss'
//       Long time with format string:     10:30:15
//       Long time with ToLongTimeString:  10:30:15
//
//       Current culture: de-DE
//       Date: 28.05.2013 10:30:15
//       Long time pattern: 'HH:mm:ss'
//       Long time with format string:     10:30:15
//       Long time with ToLongTimeString:  10:30:15
```

Remarks

The value of the current DateTime object is formatted using the pattern defined by the DateTimeFormatInfo.LongTimePattern property associated with the current culture. The return value is identical to the value returned by specifying the "T" standard date and time format string with the ToString(String) method.

ⓘ **Note**

The string returned by the **ToLongTimeString** method is culture-sensitive. It reflects the pattern defined by the current culture's **DateTimeFormatInfo.LongTimePattern** property. For example, for the en-US culture, the standard long time pattern is "h:mm:ss tt"; for the de-DE culture, it is "HH:mm:ss"; for the ja-JP culture, it is "H:mm:ss". Note that its value can vary depending on the .NET implementation and its version, the operating system and its version, and user customization.

For more information about the current thread culture, see the CultureInfo.CurrentCulture and Thread.CurrentCulture properties. For more information about format characters, format patterns, and the output they produce, see Standard date and time format strings and Custom date and time format strings. For more information about changing the format pattern associated with a format character, see the DateTimeFormatInfo class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- CultureInfo
- DateTimeFormat
- DateTimeFormatInfo
- ToString()
- Formatting Types in .NET
- Custom Date and Time Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTime.ToOADate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to the equivalent OLE Automation date.

```
C#
public double ToOADate ();
```

Returns

Double

A double-precision floating-point number that contains an OLE Automation date equivalent to the value of this instance.

Exceptions

OverflowException

The value of this instance cannot be represented as an OLE Automation Date.

Remarks

An OLE Automation date is implemented as a floating-point number whose integral component is the number of days before or after midnight, 30 December 1899, and whose fractional component represents the time on that day divided by 24. For example, midnight, 31 December 1899 is represented by 1.0; 6 A.M., 1 January 1900 is represented by 2.25; midnight, 29 December 1899 is represented by -1.0; and 6 A.M., 29 December 1899 is represented by -1.25.

The base OLE Automation Date is midnight, 30 December 1899. The minimum OLE Automation date is midnight, 1 January 0100. The maximum OLE Automation Date is the same as DateTime.MaxValue, the last moment of 31 December 9999.

The ToOADate method throws an OverflowException if the current instance represents a date that is later than MaxValue and earlier than midnight on January1, 0100. However, if the value of the current instance is MinValue, the method returns 0.

For more information, see Automation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Double
- FromOADate(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToShortDateString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to its equivalent short date string representation.

C#

```
public string ToShortDateString ();
```

Returns

String

A string that contains the short date string representation of the current DateTime object.

Examples

The following example demonstrates the ToShortDateString method. It also shows that the result of calling the ToShortDateString method is identical to calling the DateTime.ToString(String) method with "d" as the format parameter.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
   public static void Main()
   {
      DateTime dateToDisplay = new DateTime(2009, 6, 1, 8, 42, 50);
      CultureInfo originalCulture = Thread.CurrentThread.CurrentCulture;
      // Change culture to en-US.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
      Console.WriteLine("Displaying short date for {0} culture:",
```

```
                        Thread.CurrentThread.CurrentCulture.Name);
      Console.WriteLine("   {0} (Short Date String)",
                        dateToDisplay.ToShortDateString());
      // Display using 'd' standard format specifier to illustrate it is
      // identical to the string returned by ToShortDateString.
      Console.WriteLine("   {0} ('d' standard format specifier)",
                        dateToDisplay.ToString("d"));
      Console.WriteLine();

      // Change culture to fr-FR.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-FR");
      Console.WriteLine("Displaying short date for {0} culture:",
                        Thread.CurrentThread.CurrentCulture.Name);
      Console.WriteLine("   {0}", dateToDisplay.ToShortDateString());
      Console.WriteLine();

      // Change culture to nl-NL.
      Thread.CurrentThread.CurrentCulture = new CultureInfo("nl-NL");
      Console.WriteLine("Displaying short date for {0} culture:",
                        Thread.CurrentThread.CurrentCulture.Name);
      Console.WriteLine("   {0}", dateToDisplay.ToShortDateString());

      // Restore original culture.
      Thread.CurrentThread.CurrentCulture = originalCulture;
   }
}
// The example displays the following output:
//       Displaying short date for en-US culture:
//          6/1/2009 (Short Date String)
//          6/1/2009 ('d' standard format specifier)
//
//       Displaying short date for fr-FR culture:
//          01/06/2009
//
//       Displaying short date for nl-NL culture:
//          1-6-2009
```

Remarks

The value of the current DateTime object is formatted using the pattern defined by the DateTimeFormatInfo.ShortDatePattern property associated with the current culture. The return value is identical to the value returned by specifying the "d" standard DateTime format string with the ToString(String) method.

> ⓘ **Note**
>
> The string returned by the **ToShortDateString** method is culture-sensitive. It reflects the pattern defined by the current culture's **DateTimeFormatInfo.ShortDatePattern** property. For example, for the en-US culture, the standard short date pattern is

"M/d/yyyy"; for the de-DE culture, it is "dd.MM.yyyy"; for the ja-JP culture, it is "yyyy/MM/dd". Note that its value can vary depending on the .NET implementation and its version, the operating system and its version, and user customization.

For more information about the current thread culture, see the CultureInfo.CurrentCulture and Thread.CurrentCulture properties. For more information about format characters, format patterns, and the output they produce, see Standard date and time format strings and Custom date and time format strings. For more information about changing the format pattern associated with a format character, see the DateTimeFormatInfo class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- CultureInfo
- DateTimeFormat
- DateTimeFormatInfo
- ToString()
- Formatting Types in .NET
- Custom Date and Time Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToShortTimeString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to its equivalent short time string representation.

C#

```
public string ToShortTimeString ();
```

Returns

String

A string that contains the short time string representation of the current DateTime object.

Examples

The following example demonstrates the ToShortTimeString method.

C#

```
using System;
using System.Globalization;

class Sample
{
    public static void Main()
    {
        // Initialize a DateTime object.
        Console.WriteLine("Initialize the DateTime object to May 16, 2001
3:02:15 AM.\n");
        DateTime dateAndTime = new System.DateTime(2001, 5, 16, 3, 2, 15);

        // Display the name of the current culture.
        Console.WriteLine($"Current culture: \"
{CultureInfo.CurrentCulture.Name}\"\n");
        var dtfi = CultureInfo.CurrentCulture.DateTimeFormat;
```

```
        // Display the long date pattern and string.
        Console.WriteLine($"Long date pattern: \"{dtfi.LongDatePattern}\"");
        Console.WriteLine($"Long date string:  \"
{dateAndTime.ToLongDateString()}\"\n");

        // Display the long time pattern and string.
        Console.WriteLine($"Long time pattern: \"{dtfi.LongTimePattern}\"");
        Console.WriteLine($"Long time string:  \"
{dateAndTime.ToLongTimeString()}\"\n");

        // Display the short date pattern and string.
        Console.WriteLine($"Short date pattern: \"
{dtfi.ShortDatePattern}\"");
        Console.WriteLine($"Short date string:  \"
{dateAndTime.ToShortDateString()}\"\n");

        // Display the short time pattern and string.
        Console.WriteLine($"Short time pattern: \"
{dtfi.ShortTimePattern}\"");
        Console.WriteLine($"Short time string:  \"
{dateAndTime.ToShortTimeString()}\"\n");
    }
}
// The example displays output similar to the following:
//        Current culture: "en-US"
//
//        Long date pattern: "dddd, MMMM d, yyyy"
//        Long date string:  "Wednesday, May 16, 2001"
//
//        Long time pattern: "h:mm:ss tt"
//        Long time string:  "3:02:15 AM"
//
//        Short date pattern: "M/d/yyyy"
//        Short date string:  "5/16/2001"
//
//        Short time pattern: "h:mm tt"
//        Short time string:  "3:02 AM"
```

Remarks

The value of the current DateTime object is formatted using the pattern defined by the DateTimeFormatInfo.ShortTimePattern property associated with the current culture. The return value is identical to the value returned by specifying the "t" standard DateTime format string with the ToString(String) method.

> ⓘ **Note**

The string returned by the **ToShortTimeString** method is culture-sensitive. It reflects the pattern defined by the current culture's **DateTimeFormatInfo.ShortTimePattern** property. For example, for the en-US culture, the standard short time pattern is "h:mm tt"; for the de-DE culture, it is "HH:mm"; for the ja-JP culture, it is "H:mm". Note that its value can vary depending on the .NET implementation and its version, the operating system and its version, and user customization.

For more information about the current thread culture, see the CultureInfo.CurrentCulture and Thread.CurrentCulture properties. For more information about format characters, format patterns, and the output they produce, see Standard date and time format strings and Custom date and time format strings. For more information about changing the format pattern associated with a format character, see the DateTimeFormatInfo class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- CultureInfo
- DateTimeFormat
- DateTimeFormatInfo
- ToString()
- Formatting Types in .NET
- Custom Date and Time Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTime.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the value of the current DateTime object to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the value of the current DateTime object to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString()	Converts the value of the current DateTime object to its equivalent string representation using the formatting conventions of the current culture.
ToString(String, IFormatProvider)	Converts the value of the current DateTime object to its equivalent string representation using the specified format and culture-specific format information.

ToString(IFormatProvider)

Converts the value of the current DateTime object to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of value of the current DateTime object as specified by `provider`.

Implements

ToString(IFormatProvider)

Exceptions

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by `provider`.

Examples

The following example displays the string representation of a date and time using CultureInfo objects that represent five different cultures.

C#

```
using System;
using System.Globalization;

public class ToStringExample3
{
   public static void Main2()
   {
      CultureInfo[] cultures = new CultureInfo[]
{CultureInfo.InvariantCulture,
                                                    new CultureInfo("en-
us"),
                                                    new CultureInfo("fr-
fr"),
                                                    new CultureInfo("de-
DE"),
                                                    new CultureInfo("es-
ES"),
                                                    new CultureInfo("ja-
```

```
JP")};

      DateTime thisDate = new DateTime(2009, 5, 1, 9, 0, 0);

      foreach (CultureInfo culture in cultures)
      {
         string cultureName;
         if (string.IsNullOrEmpty(culture.Name))
            cultureName = culture.NativeName;
         else
            cultureName = culture.Name;

         Console.WriteLine("In {0}, {1}",
                           cultureName, thisDate.ToString(culture));
      }
      }
}
// The example produces the following output:
//    In Invariant Language (Invariant Country), 05/01/2009 09:00:00
//    In en-US, 5/1/2009 9:00:00 AM
//    In fr-FR, 01/05/2009 09:00:00
//    In de-DE, 01.05.2009 09:00:00
//    In es-ES, 01/05/2009 9:00:00
//    In ja-JP, 2009/05/01 9:00:00
```

Remarks

The value of the current DateTime object is formatted using the general date and time format specifier ('G'), which formats output using the short date pattern and the long time pattern.

The format of the short date and long time pattern is defined by the `provider` parameter. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom object that implements the IFormatProvider interface. Its GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the DateTimeFormatInfo object associated with the current culture is used. For more information, see CultureInfo.CurrentCulture.

Notes to Callers

The ToString(IFormatProvider) method returns the string representation of the date and time in the calendar used by the culture represented by the `provider` parameter. Its calendar is defined by the Calendar property. If the value of the current DateTime instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the JapaneseCalendar class.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo jaJP = new CultureInfo("ja-JP");
      jaJP.DateTimeFormat.Calendar = new JapaneseCalendar();
      DateTime date1 = new DateTime(1867, 1, 1);

      try {
         Console.WriteLine(date1.ToString(jaJP));
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("{0:d} is earlier than {1:d} or later than
{2:d}",
                           date1,

jaJP.DateTimeFormat.Calendar.MinSupportedDateTime,

jaJP.DateTimeFormat.Calendar.MaxSupportedDateTime);
      }
   }
}
// The example displays the following output:
//    1/1/1867 is earlier than 9/8/1868 or later than 12/31/9999   }
```

See also

- DateTimeFormatInfo
- CultureInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the value of the current DateTime object to its equivalent string representation using the specified format and the formatting conventions of the current culture.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom date and time format string.

Returns

String

A string representation of value of the current DateTime object as specified by `format`.

Exceptions

FormatException

The length of `format` is 1, and it is not one of the format specifier characters defined for DateTimeFormatInfo.

-or-

`format` does not contain a valid custom format pattern.

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by the current culture.

Examples

The following example uses each of the standard date and time format strings and a selection of custom date and time format strings to display the string representation of a DateTime value. The thread current culture for the example is en-US.

C#

```
using System;

public class DateToStringExample2
{
   public static void Main()
   {
      DateTime dateValue = new DateTime(2008, 6, 15, 21, 15, 07);
      // Create an array of standard format strings.
      string[] standardFmts = {"d", "D", "f", "F", "g", "G", "m", "o",
                               "R", "s", "t", "T", "u", "U", "y"};
      // Output date and time using each standard format string.
      foreach (string standardFmt in standardFmts)
         Console.WriteLine("{0}: {1}", standardFmt,
                           dateValue.ToString(standardFmt));
      Console.WriteLine();

      // Create an array of some custom format strings.
      string[] customFmts = {"h:mm:ss.ff t", "d MMM yyyy", "HH:mm:ss.f",
                             "dd MMM HH:mm:ss", @"\Mon\t\h\: M",
"HH:mm:ss.ffffzzz" };
      // Output date and time using each custom format string.
      foreach (string customFmt in customFmts)
         Console.WriteLine("'{0}': {1}", customFmt,
                           dateValue.ToString(customFmt));
   }
}
// This example displays the following output to the console:
//       d: 6/15/2008
//       D: Sunday, June 15, 2008
//       f: Sunday, June 15, 2008 9:15 PM
//       F: Sunday, June 15, 2008 9:15:07 PM
//       g: 6/15/2008 9:15 PM
//       G: 6/15/2008 9:15:07 PM
//       m: June 15
//       o: 2008-06-15T21:15:07.0000000
//       R: Sun, 15 Jun 2008 21:15:07 GMT
```

```
//       s: 2008-06-15T21:15:07
//       t: 9:15 PM
//       T: 9:15:07 PM
//       u: 2008-06-15 21:15:07Z
//       U: Monday, June 16, 2008 4:15:07 AM
//       y: June, 2008
//
//       'h:mm:ss.ff t': 9:15:07.00 P
//       'd MMM yyyy': 15 Jun 2008
//       'HH:mm:ss.f': 21:15:07.0
//       'dd MMM HH:mm:ss': 15 Jun 21:15:07
//       '\Mon\t\h\: M': Month: 6
//       'HH:mm:ss.ffffzzz': 21:15:07.0000-07:00
```

Remarks

The ToString(String) method returns the string representation of a date and time value in a specific format that uses the formatting conventions of the current culture; for more information, see CultureInfo.CurrentCulture.

The `format` parameter should contain either a single format specifier character (see Standard Date and Time Format Strings) or a custom format pattern (see Custom Date and Time Format Strings) that defines the format of the returned string. If `format` is `null` or an empty string, the general format specifier, 'G', is used.

Some uses of this method include:

- Getting a string that displays the date and time in the current culture's short date and time format. To do this, you use the "G" format specifier.
- Getting a string that contains only the month and year. To do this, you use the "MM/yyyy" format string. The format string uses the current culture's date separator.
- Getting a string that contains the date and time in a specific format. For example, the "MM/dd/yyyyHH:mm" format string displays the date and time string in a fixed format such as "19//03//2013 18:06". The format string uses "/" as a fixed date separator regardless of culture-specific settings.
- Getting a date in a condensed format that could be used for serializing a date string. For example, the "yyyyMMdd" format string displays a four-digit year followed by a two-digit month and a two-digit day with no date separator.

The following example uses these three format strings to display a date and time value by using the conventions of the en-US and fr-FR cultures.

C#

```
using System;
using System.Globalization;

public class ToStringExample5
{
    public static void Main3()
    {
        string[] formats = { "G", "MM/yyyy", @"MM\/dd\/yyyy HH:mm",
"yyyyMMdd" };
        string[] cultureNames = { "en-US", "fr-FR" };
        DateTime date = new DateTime(2015, 8, 18, 13, 31, 17);
        foreach (var cultureName in cultureNames)
        {
            var culture = new CultureInfo(cultureName);
            CultureInfo.CurrentCulture = culture;
            Console.WriteLine(culture.NativeName);
            foreach (var format in formats)
                Console.WriteLine($"   {format}:
{date.ToString(format)}");
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//       English (United States)
//          G: 8/18/2015 1:31:17 PM
//          MM/yyyy: 08/2015
//          MM\/dd\/yyyy HH:mm: 08/18/2015 13:31
//          yyyyMMdd: 20150818
//
//       français (France)
//          G: 18/08/2015 13:31:17
//          MM/yyyy: 08/2015
//          MM\/dd\/yyyy HH:mm: 08/18/2015 13:31
//          yyyyMMdd: 20150818
```

Notes to Callers

The ToString(String) method returns the string representation of the date and time in the calendar used by the current culture. If the value of the current DateTime instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the HebrewCalendar class when the current culture is Hebrew (Israel).

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example3
{
    public static void Main()
    {
        DateTime date1 = new DateTime(1550, 7, 21);
        CultureInfo dft;
        CultureInfo heIL = new CultureInfo("he-IL");
        heIL.DateTimeFormat.Calendar = new HebrewCalendar();

        // Change current culture to he-IL.
        dft = Thread.CurrentThread.CurrentCulture;
        Thread.CurrentThread.CurrentCulture = heIL;

        // Display the date using the current culture's calendar.
        try
        {
            Console.WriteLine(date1.ToString("G"));
        }
        catch (ArgumentOutOfRangeException)
        {
            Console.WriteLine("{0} is earlier than {1} or later than
{2}",
                              date1.ToString("d",
CultureInfo.InvariantCulture),

heIL.DateTimeFormat.Calendar.MinSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture),

heIL.DateTimeFormat.Calendar.MaxSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture));
        }

        // Restore the default culture.
        Thread.CurrentThread.CurrentCulture = dft;
    }
}
// The example displays the following output:
//    07/21/1550 is earlier than 01/01/1583 or later than 09/29/2239
```

See also

- DateTimeFormatInfo
- CultureInfo
- Formatting Types in .NET
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the value of the current DateTime object to its equivalent string representation using the formatting conventions of the current culture.

C#

```
public override string ToString ();
```

Returns

String

A string representation of the value of the current DateTime object.

Exceptions

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by the current culture.

Examples

The following example illustrates how the string representation of a DateTime value returned by the ToString() method depends on the thread current culture. It changes the current culture to en-US, fr-FR, and ja-JP, and in each case calls the ToString()

method to return the string representation of a date and time value using that culture.

C#

```
using System;
using System.Globalization;

public class ToStringExample1
{
    public static void Main()
    {
        CultureInfo currentCulture = CultureInfo.CurrentCulture;
        DateTime exampleDate = new DateTime(2021, 5, 1, 18, 32, 6);

        // Change the current culture to en-US and display the date.
        CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
        Console.WriteLine(exampleDate.ToString());

        // Change the current culture to fr-FR and display the date.
        CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("fr-FR");
        Console.WriteLine(exampleDate.ToString());

        // Change the current culture to ja-JP and display the date.
        CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("ja-JP");
        Console.WriteLine(exampleDate.ToString());

        // Restore the original culture
        CultureInfo.CurrentCulture = currentCulture;
    }
}

// The example displays the following output to the console:
//       5/1/2021 6:32:06 PM
//       01/05/2021 18:32:06
//       2021/05/01 18:32:06
```

Remarks

The value of the current DateTime object is formatted using the general date and time format specifier ('G'). To format it using a specific date and time format specifier, call the ToString(String) method. To format it using the general date and time format specifier ('G') for a specific culture, call the ToString(IFormatProvider) method. To format it using a specific date and time format specifier and the conventions of a specific culture, call the ToString(String, IFormatProvider) method.

This method uses formatting information derived from the current culture. In particular, it combines the custom format strings returned by the ShortDatePattern and LongTimePattern properties of the DateTimeFormatInfo object returned by the

`Thread.CurrentThread.CurrentCulture.DateTimeFormat` property. For more information, see CultureInfo.CurrentCulture. Other overloads of the ToString method enable you to specify the culture whose formatting to use and to define the output pattern of the DateTime value.

Notes to Callers

The ToString() method returns the string representation of the date and time in the calendar used by the current culture. If the value of the current DateTime instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the HijriCalendar class when the current culture is Arabic (Syria).

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example2
{
    public static void Main()
    {
        DateTime date1 = new DateTime(550, 1, 1);
        CultureInfo dft;
        CultureInfo arSY = new CultureInfo("ar-SY");
        arSY.DateTimeFormat.Calendar = new HijriCalendar();

        // Change current culture to ar-SY.
        dft = Thread.CurrentThread.CurrentCulture;
        Thread.CurrentThread.CurrentCulture = arSY;

        // Display the date using the current culture's calendar.
        try
        {
            Console.WriteLine(date1.ToString());
        }
        catch (ArgumentOutOfRangeException)
        {
            Console.WriteLine("{0} is earlier than {1} or later than
{2}",
                              date1.ToString("d",
CultureInfo.InvariantCulture),

arSY.DateTimeFormat.Calendar.MinSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture),

arSY.DateTimeFormat.Calendar.MaxSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture));
```

```
        }

        // Restore the default culture.
        Thread.CurrentThread.CurrentCulture = dft;
    }
}
// The example displays the following output:
//    01/01/0550 is earlier than 07/18/0622 or later than 12/31/9999
```

See also

- DateTimeFormatInfo
- CultureInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the value of the current DateTime object to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A standard or custom date and time format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of value of the current DateTime object as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The length of `format` is 1, and it is not one of the format specifier characters defined for DateTimeFormatInfo.

-or-

`format` does not contain a valid custom format pattern.

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by `provider`.

Examples

The following example uses each of the standard date time format strings to display the string representation of a date and time for four different cultures.

C#

```
using System;
using System.Globalization;

public class ToStringExample4
{
   public static void Main1()
   {
      // Create an array of all supported standard date and time format
specifiers.
      string[] formats = {"d", "D", "f", "F", "g", "G", "m", "o", "r",
```

```
                              "s", "t", "T", "u", "U", "Y"};
      // Create an array of four cultures.
      CultureInfo[] cultures = {CultureInfo.GetCultureInfo("de-DE"),
                                CultureInfo.GetCultureInfo("en-US"),
                                CultureInfo.GetCultureInfo("es-ES"),
                                CultureInfo.GetCultureInfo("fr-FR")};
       // Define date to be displayed.
      DateTime dateToDisplay = new DateTime(2008, 10, 31, 17, 4, 32);

      // Iterate each standard format specifier.
      foreach (string formatSpecifier in formats)
      {
         foreach (CultureInfo culture in cultures)
            Console.WriteLine("{0} Format Specifier {1, 10} Culture {2,
40}",
                              formatSpecifier, culture.Name,
                              dateToDisplay.ToString(formatSpecifier,
culture));
         Console.WriteLine();
      }
   }
}

// The example displays the following output:
//    d Format Specifier      de-DE Culture
31.10.2008
//    d Format Specifier      en-US Culture
10/31/2008
//    d Format Specifier      es-ES Culture
31/10/2008
//    d Format Specifier      fr-FR Culture
31/10/2008
//
//    D Format Specifier      de-DE Culture                Freitag, 31.
Oktober 2008
//    D Format Specifier      en-US Culture                 Friday,
October 31, 2008
//    D Format Specifier      es-ES Culture           viernes, 31 de
octubre de 2008
//    D Format Specifier      fr-FR Culture                 vendredi 31
octobre 2008
//
//    f Format Specifier      de-DE Culture          Freitag, 31. Oktober
2008 17:04
//    f Format Specifier      en-US Culture         Friday, October 31,
2008 5:04 PM
//    f Format Specifier      es-ES Culture     viernes, 31 de octubre de
2008 17:04
//    f Format Specifier      fr-FR Culture           vendredi 31 octobre
2008 17:04
//
//    F Format Specifier      de-DE Culture       Freitag, 31. Oktober
2008 17:04:32
//    F Format Specifier      en-US Culture      Friday, October 31, 2008
5:04:32 PM
```

```
//    F Format Specifier      es-ES Culture  viernes, 31 de octubre de
2008 17:04:32
//    F Format Specifier      fr-FR Culture        vendredi 31 octobre
2008 17:04:32
//
//    g Format Specifier      de-DE Culture
31.10.2008 17:04
//    g Format Specifier      en-US Culture
10/31/2008 5:04 PM
//    g Format Specifier      es-ES Culture
31/10/2008 17:04
//    g Format Specifier      fr-FR Culture
31/10/2008 17:04
//
//    G Format Specifier      de-DE Culture
31.10.2008 17:04:32
//    G Format Specifier      en-US Culture                    10/31/2008
5:04:32 PM
//    G Format Specifier      es-ES Culture
31/10/2008 17:04:32
//    G Format Specifier      fr-FR Culture
31/10/2008 17:04:32
//
//    m Format Specifier      de-DE Culture
31. Oktober
//    m Format Specifier      en-US Culture
October 31
//    m Format Specifier      es-ES Culture                           31
de octubre
//    m Format Specifier      fr-FR Culture
31 octobre
//
//    o Format Specifier      de-DE Culture              2008-10-
31T17:04:32.0000000
//    o Format Specifier      en-US Culture              2008-10-
31T17:04:32.0000000
//    o Format Specifier      es-ES Culture              2008-10-
31T17:04:32.0000000
//    o Format Specifier      fr-FR Culture              2008-10-
31T17:04:32.0000000
//
//    r Format Specifier      de-DE Culture            Fri, 31 Oct 2008
17:04:32 GMT
//    r Format Specifier      en-US Culture            Fri, 31 Oct 2008
17:04:32 GMT
//    r Format Specifier      es-ES Culture            Fri, 31 Oct 2008
17:04:32 GMT
//    r Format Specifier      fr-FR Culture            Fri, 31 Oct 2008
17:04:32 GMT
//
//    s Format Specifier      de-DE Culture                      2008-10-
31T17:04:32
//    s Format Specifier      en-US Culture                      2008-10-
31T17:04:32
//    s Format Specifier      es-ES Culture                      2008-10-
```

```
31T17:04:32
//    s Format Specifier      fr-FR Culture                     2008-10-
31T17:04:32
//
//    t Format Specifier      de-DE Culture
17:04
//    t Format Specifier      en-US Culture
5:04 PM
//    t Format Specifier      es-ES Culture
17:04
//    t Format Specifier      fr-FR Culture
17:04
//
//    T Format Specifier      de-DE Culture
17:04:32
//    T Format Specifier      en-US Culture
5:04:32 PM
//    T Format Specifier      es-ES Culture
17:04:32
//    T Format Specifier      fr-FR Culture
17:04:32
//
//    u Format Specifier      de-DE Culture                     2008-10-
31 17:04:32Z
//    u Format Specifier      en-US Culture                     2008-10-
31 17:04:32Z
//    u Format Specifier      es-ES Culture                     2008-10-
31 17:04:32Z
//    u Format Specifier      fr-FR Culture                     2008-10-
31 17:04:32Z
//
//    U Format Specifier      de-DE Culture       Freitag, 31. Oktober
2008 09:04:32
//    U Format Specifier      en-US Culture      Friday, October 31, 2008
9:04:32 AM
//    U Format Specifier      es-ES Culture   viernes, 31 de octubre de
2008 9:04:32
//    U Format Specifier      fr-FR Culture        vendredi 31 octobre
2008 09:04:32
//
//    Y Format Specifier      de-DE Culture
Oktober 2008
//    Y Format Specifier      en-US Culture
October 2008
//    Y Format Specifier      es-ES Culture
octubre de 2008
//    Y Format Specifier      fr-FR Culture
octobre 2008
```

The following example demonstrates different ways of formatting a DateTime value using the invariant DateTimeFormatInfo.

```
C#
```

```
using System;
using System.Globalization;

public class MainClass
{
    public static void Main(string[] args)
    {
        DateTime dt = DateTime.Now;
        String[] format = {
            "d", "D",
            "f", "F",
            "g", "G",
            "m",
            "r",
            "s",
            "t", "T",
            "u", "U",
            "y",
            "dddd, MMMM dd yyyy",
            "ddd, MMM d \"'\"yy",
            "dddd, MMMM dd",
            "M/yy",
            "dd-MM-yy",
        };
        string date;
        for (int i = 0; i < format.Length; i++)
        {
            date = dt.ToString(format[i],
DateTimeFormatInfo.InvariantInfo);
            Console.WriteLine(string.Concat(format[i], " :", date));
        }

        /** Output.
         *
         * d :08/17/2000
         * D :Thursday, August 17, 2000
         * f :Thursday, August 17, 2000 16:32
         * F :Thursday, August 17, 2000 16:32:32
         * g :08/17/2000 16:32
         * G :08/17/2000 16:32:32
         * m :August 17
         * r :Thu, 17 Aug 2000 23:32:32 GMT
         * s :2000-08-17T16:32:32
         * t :16:32
         * T :16:32:32
         * u :2000-08-17 23:32:32Z
         * U :Thursday, August 17, 2000 23:32:32
         * y :August, 2000
         * dddd, MMMM dd yyyy :Thursday, August 17 2000
         * ddd, MMM d "'"yy :Thu, Aug 17 '00
         * dddd, MMMM dd :Thursday, August 17
         * M/yy :8/00
         * dd-MM-yy :17-08-00
         */
```

```
    }
}
```

Remarks

The `format` parameter can contain either a single format specifier character (see Standard Date and Time Format Strings) or a custom format pattern (see Custom Date and Time Format Strings). If `format` is `null` or an empty string (""), the standard format specifier, "G", is used.

The `provider` parameter defines the pattern that corresponds to the standard format specifiers, as well as the symbols and names of date and time components. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom object that implements the IFormatProvider interface. Its GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the DateTimeFormatInfo associated with the current culture is used. For more information, see CultureInfo.CurrentCulture.

Notes to Callers

The ToString(String, IFormatProvider) method returns the string representation of the date and time in the calendar used by the `provider` parameter. Its calendar is defined by the Calendar property. If the value of the current DateTime instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the UmAlQuraCalendar class.

C#

```
using System;
using System.Globalization;
```

```
public class Example4
{
    public static void Main()
    {
        CultureInfo arSA = new CultureInfo("ar-SA");
        arSA.DateTimeFormat.Calendar = new UmAlQuraCalendar();
        DateTime date1 = new DateTime(1890, 9, 10);

        try
        {
            Console.WriteLine(date1.ToString("d", arSA));
        }
        catch (ArgumentOutOfRangeException)
        {
            Console.WriteLine("{0:d} is earlier than {1:d} or later than
{2:d}",
                              date1,

arSA.DateTimeFormat.Calendar.MinSupportedDateTime,

arSA.DateTimeFormat.Calendar.MaxSupportedDateTime);
        }
    }
}
// The example displays the following output:
//    9/10/1890 is earlier than 4/30/1900 or later than 5/13/2029
```

See also

- DateTimeFormatInfo
- CultureInfo
- Formatting Types in .NET
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ToUniversalTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTime object to Coordinated Universal Time (UTC).

C#

```
public DateTime ToUniversalTime ();
```

Returns

DateTime

An object whose Kind property is Utc, and whose value is the UTC equivalent to the value of the current DateTime object, or DateTime.MaxValue if the converted value is too large to be represented by a DateTime object, or DateTime.MinValue if the converted value is too small to be represented by a DateTime object.

Examples

The following example demonstrates the ToUniversalTime method.

C#

```
using System;

class Example
{
    static void Main()
    {
        DateTime localDateTime, univDateTime;

        Console.WriteLine("Enter a date and time.");
        string strDateTime = Console.ReadLine();

        try {
            localDateTime = DateTime.Parse(strDateTime);
            univDateTime = localDateTime.ToUniversalTime();
```

```
            Console.WriteLine("{0} local time is {1} universal time.",
                                localDateTime,
                                    univDateTime);
        }
        catch (FormatException) {
            Console.WriteLine("Invalid format.");
            return;
        }

        Console.WriteLine("Enter a date and time in universal time.");
        strDateTime = Console.ReadLine();

        try {
            univDateTime = DateTime.Parse(strDateTime);
            localDateTime = univDateTime.ToLocalTime();

            Console.WriteLine("{0} universal time is {1} local time.",
                                univDateTime,
                                localDateTime);
        }
        catch (FormatException) {
            Console.WriteLine("Invalid format.");
            return;
        }
    }
}
// The example displays output like the following when run on a
// computer whose culture is en-US in the Pacific Standard Time zone:
//     Enter a date and time.
//     12/10/2015 6:18 AM
//     12/10/2015 6:18:00 AM local time is 12/10/2015 2:18:00 PM universal
time.
//     Enter a date and time in universal time.
//     12/20/2015 6:42:00
//     12/20/2015 6:42:00 AM universal time is 12/19/2015 10:42:00 PM local
time.
```

The following example uses the SpecifyKind method to demonstrate how the Kind property influences the ToLocalTime and ToUniversalTime conversion methods.

C#

```
// This code example demonstrates the DateTime Kind, Now, and
// UtcNow properties, and the SpecifyKind(), ToLocalTime(),
// and ToUniversalTime() methods.

using System;

class Sample
{
    public static void Main()
    {
        // Get the date and time for the current moment, adjusted
```

```
        // to the local time zone.

        DateTime saveNow = DateTime.Now;

        // Get the date and time for the current moment expressed
        // as coordinated universal time (UTC).

        DateTime saveUtcNow = DateTime.UtcNow;
        DateTime myDt;

        // Display the value and Kind property of the current moment
        // expressed as UTC and local time.

        DisplayNow("UtcNow: ..........", saveUtcNow);
        DisplayNow("Now: ............", saveNow);
        Console.WriteLine();

        // Change the Kind property of the current moment to
        // DateTimeKind.Utc and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Utc);
        Display("Utc: ............", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Local and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Local);
        Display("Local: ...........", myDt);

        // Change the Kind property of the current moment to
        // DateTimeKind.Unspecified and display the result.

        myDt = DateTime.SpecifyKind(saveNow, DateTimeKind.Unspecified);
        Display("Unspecified: .....", myDt);
    }

    // Display the value and Kind property of a DateTime structure, the
    // DateTime structure converted to local time, and the DateTime
    // structure converted to universal time.

    public static string datePatt = @"M/d/yyyy hh:mm:ss tt";
    public static void Display(string title, DateTime inputDt)
    {
        DateTime dispDt = inputDt;
        string dtString;

        // Display the original DateTime.

        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, dispDt.Kind);

        // Convert inputDt to local time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is not
```

```
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was universal time.

        dispDt = inputDt.ToLocalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToLocalTime:     {0}, Kind = {1}",
                          dtString, dispDt.Kind);

        // Convert inputDt to universal time and display the result.
        // If inputDt.Kind is DateTimeKind.Utc, the conversion is not
performed.
        // If inputDt.Kind is DateTimeKind.Local, the conversion is
performed.
        // If inputDt.Kind is DateTimeKind.Unspecified, the conversion is
        // performed as if inputDt was local time.

        dispDt = inputDt.ToUniversalTime();
        dtString = dispDt.ToString(datePatt);
        Console.WriteLine("  ToUniversalTime: {0}, Kind = {1}",
                          dtString, dispDt.Kind);
        Console.WriteLine();
    }

    // Display the value and Kind property for DateTime.Now and
DateTime.UtcNow.

    public static void DisplayNow(string title, DateTime inputDt)
    {
        string dtString = inputDt.ToString(datePatt);
        Console.WriteLine("{0} {1}, Kind = {2}",
                          title, dtString, inputDt.Kind);
    }
}

/*
This code example produces the following results:

UtcNow: .......... 5/6/2005 09:34:42 PM, Kind = Utc
Now: ............. 5/6/2005 02:34:42 PM, Kind = Local

Utc: ............. 5/6/2005 02:34:42 PM, Kind = Utc
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 02:34:42 PM, Kind = Utc

Local: ........... 5/6/2005 02:34:42 PM, Kind = Local
  ToLocalTime:     5/6/2005 02:34:42 PM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

Unspecified: ..... 5/6/2005 02:34:42 PM, Kind = Unspecified
  ToLocalTime:     5/6/2005 07:34:42 AM, Kind = Local
  ToUniversalTime: 5/6/2005 09:34:42 PM, Kind = Utc

*/
```

Remarks

The Coordinated Universal Time (UTC) is equal to the local time minus the UTC offset. For more information about the UTC offset, see TimeZoneInfo.GetUtcOffset. The conversion also takes into account the daylight saving time rule that applies to the time represented by the current DateTime object.

> **Important**
>
> On Windows XP systems, the **ToUniversalTime** method recognizes only the current adjustment rule when converting from local time to UTC. As a result, conversions for periods before the current adjustment rule came into effect may not accurately reflect the difference between local time and UTC.

Starting with the .NET Framework version 2.0, the value returned by the ToUniversalTime method is determined by the Kind property of the current DateTime object. The following table describes the possible results.

Expand table

Kind	Results
Utc	No conversion is performed.
Local	The current DateTime object is converted to UTC.
Unspecified	The current DateTime object is assumed to be a local time, and the conversion is performed as if Kind were Local.

> **Note**
>
> The **ToUniversalTime** method converts a **DateTime** value from local time to UTC. To convert the time in a non-local time zone to UTC, use the **TimeZoneInfo.ConvertTimeToUtc(DateTime, TimeZoneInfo)** method. To convert a time whose offset from UTC is known, use the **ToUniversalTime** method.

If the date and time instance value is an ambiguous time, this method assumes that it is a standard time. (An ambiguous time is one that can map either to a standard time or to a daylight saving time in the local time zone) If the date and time instance value is an invalid time, this method simply subtracts the local time from the local time zone's UTC offset to return UTC. (An invalid time is one that does not exist because of the application of daylight saving time adjustment rules.)

Notes to Callers

The ToUniversalTime() method is sometimes used to convert a local time to UTC. The ToLocalTime() method is then called to restore the original local time. However, if the original time represents an invalid time in the local time zone, the two local time values will not be equal. For additional information and an example, see the ToLocalTime() method.

On Windows XP systems, the ToUniversalTime() method recognizes only the current adjustment rule for the local time zone, which it applies to all dates, including down-level dates (that is, dates that are earlier than the starting date of the current adjustment rule). Applications running on Windows XP that require historically accurate local date and time calculations must work around this behavior by using the FindSystemTimeZoneById(String) method to retrieve a TimeZoneInfo object that corresponds to the local time zone and calling its ConvertTimeToUtc(DateTime, TimeZoneInfo) method.

The following example illustrates the difference between the ToUniversalTime() and ConvertTimeToUtc(DateTime, TimeZoneInfo) methods on a Windows XP system in the U.S. Pacific Time zone. The first two method calls apply the current time zone adjustment rule (which went into effect in 2007) to a date in 2006. The current adjustment rule provides for the transition to daylight saving time on the second Sunday of March; the previous rule, which was in effect in 2006, provided for the transition to daylight saving time to occur on the first Sunday of April. Only the third method call accurately performs this historical date and time conversion.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTime date1 = new DateTime(2006, 3, 21, 2, 0, 0);

      Console.WriteLine(date1.ToUniversalTime());
      Console.WriteLine(TimeZoneInfo.ConvertTimeToUtc(date1));

      TimeZoneInfo tz = TimeZoneInfo.FindSystemTimeZoneById("Pacific
Standard Time");
      Console.WriteLine(TimeZoneInfo.ConvertTimeToUtc(date1, tz));
   }
}
// The example displays the following output on Windows XP systems:
//       3/21/2006 9:00:00 AM
```

```
//        3/21/2006 9:00:00 AM
//        3/21/2006 10:00:00 AM
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToLocalTime()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current datetime instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current datetime instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTime.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, DateTime)	Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTime)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, DateTime)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)	Converts the span representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.

Remarks

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

TryParse(ReadOnlySpan<Char>, DateTime)

Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out DateTime result);
```

Parameters

`s` ReadOnlySpan<Char>

A string containing a date and time to convert.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if the `s` parameter is `null`, is an empty string (""), or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, DateTime)

Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, out DateTime result);
```

Parameters

`s` String

A string containing a date and time to convert.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if the `s` parameter is `null`, is an empty string (""), or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Examples

The following example passes a number of date and time strings to the DateTime.TryParse(String, DateTime) method.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] dateStrings = {"05/01/2009 14:57:32.8", "2009-05-01
14:57:32.8",
                              "2009-05-01T14:57:32.8375298-04:00",
"5/01/2008",
                              "5/01/2008 14:57:32.80 -07:00",
                              "1 May 2008 2:57:32.8 PM", "16-05-2009
1:00:32 PM",
                              "Fri, 15 May 2009 20:10:57 GMT" };
      DateTime dateValue;

      Console.WriteLine("Attempting to parse strings using {0} culture.",
                        CultureInfo.CurrentCulture.Name);
      foreach (string dateString in dateStrings)
      {
         if (DateTime.TryParse(dateString, out dateValue))
            Console.WriteLine("  Converted '{0}' to {1} ({2}).",
dateString,
                              dateValue, dateValue.Kind);
         else
            Console.WriteLine("  Unable to parse '{0}'.", dateString);
      }
   }
}
// The example displays output like the following:
//    Attempting to parse strings using en-US culture.
//      Converted '05/01/2009 14:57:32.8' to 5/1/2009 2:57:32 PM
(Unspecified).
//      Converted '2009-05-01 14:57:32.8' to 5/1/2009 2:57:32 PM
(Unspecified).
//      Converted '2009-05-01T14:57:32.8375298-04:00' to 5/1/2009
11:57:32 AM (Local).
//
//      Converted '5/01/2008' to 5/1/2008 12:00:00 AM (Unspecified).
//      Converted '5/01/2008 14:57:32.80 -07:00' to 5/1/2008 2:57:32 PM
(Local).
//      Converted '1 May 2008 2:57:32.8 PM' to 5/1/2008 2:57:32 PM
(Unspecified).
//      Unable to parse '16-05-2009 1:00:32 PM'.
//      Converted 'Fri, 15 May 2009 20:10:57 GMT' to 5/15/2009 1:10:57 PM
(Local).
```

Remarks

The DateTime.TryParse(String, DateTime) method is similar to the DateTime.Parse(String) method, except that the TryParse(String, DateTime) method does not throw an exception if the conversion fails.

The string `s` is parsed using formatting information in the current DateTimeFormatInfo object, which is supplied implicitly by the current culture.

This method tries to ignore unrecognized data, if possible, and fills in missing month, day, and year information with the current date. If `s` contains only a date and no time, this method assumes the time is 12:00 midnight. If `s` includes a date component with a two-digit year, it is converted to a year in the current culture's current calendar based on the value of the Calendar.TwoDigitYearMax property. Any leading, inner, or trailing white space character in `s` is ignored. The date and time can be bracketed with a pair of leading and trailing NUMBER SIGN characters ('#', U+0023), and can be trailed with one or more NULL characters (U+0000).

Because the DateTime.TryParse(String, DateTime) method tries to parse the string representation of a date and time using the formatting rules of the current culture, trying to parse a particular string across different cultures can either fail or return different results. If a specific date and time format will be parsed across different locales, use the DateTime.TryParse(String, IFormatProvider, DateTimeStyles, DateTime) method or one of the overloads of the TryParseExact method and provide a format specifier.

If `s` is the string representation of a leap day in a leap year in the current calendar, the method parses `s` successfully. If `s` is the string representation of a leap day in a non-leap year in the current culture's current calendar, the parse operation fails and the method returns `false`.

If `s` contains no time zone information, `result` contains a DateTime value whose Kind property is DateTimeKind.Unspecified when the method returns. If the string to be parsed contains time zone information, `result` contains a DateTime value whose Kind property is DateTimeKind.Local when the method returns.

Notes to Callers

Formatting is influenced by properties of the current DateTimeFormatInfo object, which by default are derived from the **Regional and Language Options** item in Control Panel. The TryParse method can unexpectedly fail and return `False` if the current DateSeparator and TimeSeparator properties are set to the same value.

See also

- Parse
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTime)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out DateTime result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateTime

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, DateTime)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider, out
DateTime result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateTime

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, DateTimeStyles, DateTime)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider,
System.Globalization.DateTimeStyles styles, out DateTime result);
```

Parameters

`s` String

A string containing a date and time to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date in relation to the current time zone or the current date. A typical value

to specify is None.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if the `s` parameter is `null`, is an empty string (""), or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`styles` is not a valid DateTimeStyles value.

-or-

`styles` contains an invalid combination of DateTimeStyles values (for example, both AssumeLocal and AssumeUniversal).

NotSupportedException

`provider` is a neutral culture and cannot be used in a parsing operation.

Examples

The following example illustrates the DateTime.TryParse(String, IFormatProvider, DateTimeStyles, DateTime) method.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string dateString;
      CultureInfo culture;
      DateTimeStyles styles;
```

```
      DateTime dateResult;

      // Parse a date and time with no styles.
      dateString = "03/01/2009 10:00 AM";
      culture = CultureInfo.CreateSpecificCulture("en-US");
      styles = DateTimeStyles.None;
      if (DateTime.TryParse(dateString, culture, styles, out dateResult))
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, dateResult, dateResult.Kind);
      else
         Console.WriteLine("Unable to convert {0} to a date and time.",
                           dateString);

      // Parse the same date and time with the AssumeLocal style.
      styles = DateTimeStyles.AssumeLocal;
      if (DateTime.TryParse(dateString, culture, styles, out dateResult))
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, dateResult, dateResult.Kind);
      else
         Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);

      // Parse a date and time that is assumed to be local.
      // This time is five hours behind UTC. The local system's time zone
is
      // eight hours behind UTC.
      dateString = "2009/03/01T10:00:00-5:00";
      styles = DateTimeStyles.AssumeLocal;
      if (DateTime.TryParse(dateString, culture, styles, out dateResult))
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, dateResult, dateResult.Kind);
      else
         Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);

      // Attempt to convert a string in improper ISO 8601 format.
      dateString = "03/01/2009T10:00:00-5:00";
      if (DateTime.TryParse(dateString, culture, styles, out dateResult))
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, dateResult, dateResult.Kind);
      else
         Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);

      // Assume a date and time string formatted for the fr-FR culture is
the local
      // time and convert it to UTC.
      dateString = "2008-03-01 10:00";
      culture = CultureInfo.CreateSpecificCulture("fr-FR");
      styles = DateTimeStyles.AdjustToUniversal |
DateTimeStyles.AssumeLocal;
      if (DateTime.TryParse(dateString, culture, styles, out dateResult))
         Console.WriteLine("{0} converted to {1} {2}.",
                           dateString, dateResult, dateResult.Kind);
      else
```

```
            Console.WriteLine("Unable to convert {0} to a date and time.",
dateString);
        }
    }
// The example displays the following output to the console:
//       03/01/2009 10:00 AM converted to 3/1/2009 10:00:00 AM
Unspecified.
//       03/01/2009 10:00 AM converted to 3/1/2009 10:00:00 AM Local.
//       2009/03/01T10:00:00-5:00 converted to 3/1/2009 7:00:00 AM Local.
//       Unable to convert 03/01/2009T10:00:00-5:00 to a date and time.
//       2008-03-01 10:00 converted to 3/1/2008 6:00:00 PM Utc.
```

Remarks

For more information about this API, see Supplemental API remarks for DateTime.TryParse.

Notes to Callers

Formatting is influenced by properties of the current DateTimeFormatInfo object, which is supplied by the `provider` parameter. The TryParse method can unexpectedly fail and return `False` if the current DateSeparator and TimeSeparator properties are set to the same value.

See also

- Parse
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)

Converts the span representation of a date and time to its DateTime equivalent using the specified culture-specific format information and formatting style, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, System.Globalization.DateTimeStyles styles, out DateTime
result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the date and time to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date in relation to the current time zone or the current date. A typical value to specify is None.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if the `s` parameter is `null`, is an empty string (""), or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly. The method returns a value that indicates whether the conversion succeeded.

Overloads

Expand table

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTime)	Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime)	Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly. The method returns a value that indicates whether the conversion succeeded.

Remarks

> **Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTime)

Converts the specified span representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider,
System.Globalization.DateTimeStyles style, out DateTime result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing a date and time to convert.

format ReadOnlySpan<Char>

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if either the `s` or `format` parameter is `null`, is an empty string, or does not contain a date and time that correspond to the pattern specified in `format`. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTime)

Converts the specified char span of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s, string?[]?
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style, out DateTime result);
```

Parameters

`s` ReadOnlySpan<Char>

The span containing the string to parse.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date in relation to the current time zone or the current date. A typical value to specify is None.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if the `s` parameter is `null`, is Empty, or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParseExact (string? s, string? format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
DateTime result);
```

Parameters

`s` String

A string containing a date and time to convert.

`format` String

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if either the `s` or `format` parameter is `null`, is an empty string, or does not contain a date and time that correspond to the pattern specified in `format`. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`styles` is not a valid DateTimeStyles value.

-or-

`styles` contains an invalid combination of DateTimeStyles values (for example, both AssumeLocal and AssumeUniversal).

Examples

The following example demonstrates the DateTime.TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime) method. Note that the string " 5/01/2009 8:30 AM" cannot be parsed successfully when the `styles` parameter equals DateTimeStyles.None because leading spaces are not allowed by `format`. Additionally, the string "5/01/2009 09:00" cannot be parsed successfully with a `format` of "MM/dd/yyyyhh:mm" because the date string does not precede the month number with a leading zero, as `format` requires.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo enUS = new CultureInfo("en-US");
      string dateString;
      DateTime dateValue;

      // Parse date with no style flags.
      dateString = " 5/01/2009 8:30 AM";
      if (DateTime.TryParseExact(dateString, "g", enUS,
                                 DateTimeStyles.None, out dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Allow a leading space in the date string.
      if (DateTime.TryParseExact(dateString, "g", enUS,
                                 DateTimeStyles.AllowLeadingWhite, out
dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
```

```
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Use custom formats with M and MM.
      dateString = "5/01/2009 09:00";
      if (DateTime.TryParseExact(dateString, "M/dd/yyyy hh:mm", enUS,
                                 DateTimeStyles.None, out dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Allow a leading space in the date string.
      if (DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm", enUS,
                                 DateTimeStyles.None, out dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Parse a string with time zone information.
      dateString = "05/01/2009 01:30:42 PM -05:00";
      if (DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm:ss tt
zzz", enUS,
                                 DateTimeStyles.None, out dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Allow a leading space in the date string.
      if (DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm:ss tt
zzz", enUS,
                                 DateTimeStyles.AdjustToUniversal, out
dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                           dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      // Parse a string representing UTC.
      dateString = "2008-06-11T16:11:20.0904778Z";
      if (DateTime.TryParseExact(dateString, "o",
CultureInfo.InvariantCulture,
                                    DateTimeStyles.None, out dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
```

```
dateValue,
                            dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);

      if (DateTime.TryParseExact(dateString, "o",
CultureInfo.InvariantCulture,
                                  DateTimeStyles.RoundtripKind, out
dateValue))
         Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString,
dateValue,
                            dateValue.Kind);
      else
         Console.WriteLine("'{0}' is not in an acceptable format.",
dateString);
   }
}
// The example displays the following output:
//    ' 5/01/2009 8:30 AM' is not in an acceptable format.
//    Converted ' 5/01/2009 8:30 AM' to 5/1/2009 8:30:00 AM
(Unspecified).
//    Converted '5/01/2009 09:00' to 5/1/2009 9:00:00 AM (Unspecified).
//    '5/01/2009 09:00' is not in an acceptable format.
//    Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 11:30:42 AM
(Local).
//    Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 6:30:42 PM
(Utc).
//    Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 9:11:20 AM
(Local).
//    Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 4:11:20 PM
(Utc).
```

Remarks

The DateTime.TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime) method parses the string representation of a date, which must be in the format defined by the `format` parameter. It is similar to the DateTime.ParseExact(String, String, IFormatProvider, DateTimeStyles) method, except that the TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime) method does not throw an exception if the conversion fails.

The `s` parameter contains the date and time to parse and must be in a format defined by the `format` parameter. If date, time, and time zone elements are present in `s`, they must also appear in the order specified by `format`. If `format` defines a date with no time element and the parse operation succeeds, the resulting DateTime value has a time of midnight (00:00:00). If `format` defines a time with no date element and the parse operation succeeds, the resulting DateTime value by default

has a date of `DateTime.Now.Date`, or it has a date of `DateTime.MinValue.Date` if `styles` includes the DateTimeStyles.NoCurrentDateDefault flag. The `style` parameter determines whether the `s` parameter can contain leading, inner, or trailing white space characters.

If `s` contains no time zone information, the Kind property of the returned DateTime object is DateTimeKind.Unspecified. This behavior can be changed by using the DateTimeStyles.AssumeLocal flag, which returns a DateTime value whose Kind property is DateTimeKind.Local, or by using the DateTimeStyles.AssumeUniversal and DateTimeStyles.AdjustToUniversal flags, which returns a DateTime value whose Kind property is DateTimeKind.Utc. If s contains time zone information, the time is converted to local time, if necessary, and the Kind property of the returned DateTime object is set to DateTimeKind.Local. This behavior can be changed by using the DateTimeStyles.RoundtripKind flag to not convert Coordinated Universal Time (UTC) to a local time and set the Kind property to DateTimeKind.Utc.

The `format` parameter contains a pattern that corresponds to the expected format of the `s` parameter. The pattern in the `format` parameter consists of one or more custom format specifiers from the Custom Date and Time Format Strings table, or a single standard format specifier, which identifies a predefined pattern, from the Standard Date and Time Format Strings table.

If you do not use date or time separators in a custom format pattern, use the invariant culture for the `provider` parameter and the widest form of each custom format specifier. For example, if you want to specify hours in the pattern, specify the wider form, "HH", instead of the narrower form, "H".

> **Note**
>
> Rather than requiring that `s` conform to a single format for the parse operation to succeed, you can call the **DateTime.TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime)** method and specify multiple permitted formats. This makes the parse operation more likely to succeed.

The particular date and time symbols and strings (such as the names of the days of the week in a particular language) used in `s` are defined by the `provider` parameter, as is the precise format of `s` if `format` is a standard format specifier string. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture used to interpret `s`. The DateTimeFormatInfo object returned by its DateTimeFormat property defines

the symbols and formatting in `s`.

- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom IFormatProvider implementation whose GetFormat method returns either the CultureInfo object or the DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter includes one or more members of the DateTimeStyles enumeration that determine whether and where white space not defined by `format` can appear in `s` and that control the precise behavior of the parse operation. The following table describes how each member of the DateTimeStyles enumeration affects the operation of the TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime) method.

Expand table

DateTimeStyles member	Description
AdjustToUniversal	Parses `s` and, if necessary, converts it to UTC. If `s` includes a time zone offset, or if `s` contains no time zone information but `styles` includes the DateTimeStyles.AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc. If `s` indicates that it represents UTC, or if `s` does not contain time zone information but `styles` includes the DateTimeStyles.AssumeUniversal flag, the method parses the string, performs no time zone conversion on the returned DateTime value, and sets the Kind property to DateTimeKind.Utc. In all other cases, the flag has no effect.
AllowInnerWhite	Specifies that white space not defined by `format` can appear between any individual date or time element.
AllowLeadingWhite	Specifies that white space not defined by `format` can appear at the beginning of `s`.
AllowTrailingWhite	Specifies that white space not defined by `format` can appear at the end of `s`.
AllowWhiteSpaces	Specifies that `s` may contain leading, inner, and trailing white spaces not defined by `format`.

DateTimeStyles member	Description
AssumeLocal	Specifies that if `s` lacks any time zone information, it is assumed to represent a local time. Unless the DateTimeStyles.AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local.
AssumeUniversal	Specifies that if `s` lacks any time zone information, it is assumed to represent UTC. Unless the DateTimeStyles.AdjustToUniversal flag is present, the method converts the returned DateTime value from UTC to local time and sets its Kind property to DateTimeKind.Local.
NoCurrentDateDefault	If `s` contains time without date information, the date of the return value is set to `DateTime.MinValue.Date`.
None	The `s` parameter is parsed using default values. No white space other than that present in `format` is allowed. If `s` lacks a date component, the date of the returned DateTime value is set to 1/1/0001. If `s` contains no time zone information, the Kind property of the returned DateTime object is set to DateTimeKind.Unspecified. If time zone information is present in `s`, the time is converted to local time and the Kind property of the returned DateTime object is set to DateTimeKind.Local.
RoundtripKind	For strings that contain time zone information, tries to prevent the conversion to a DateTime value with its Kind property set to DateTimeKind.Local. This flag primarily prevents the conversion of UTC times to local times.

Notes to Callers

In the .NET Framework 4, the TryParseExact method returns `false` if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- ParseExact
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime)

Converts the specified string representation of a date and time to its DateTime equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string?[]? formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
DateTime result);
```

Parameters

`s` String

A string that contains a date and time to convert.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

`result` DateTime

When this method returns, contains the DateTime value equivalent to the date and time contained in `s`, if the conversion succeeded, or DateTime.MinValue if the conversion failed. The conversion fails if `s` or `formats` is `null`, `s` or an element of `formats` is an empty string, or the format of `s` is not exactly as specified by at least one of the format patterns in `formats`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `s` parameter was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`styles` is not a valid DateTimeStyles value.

-or-

`styles` contains an invalid combination of DateTimeStyles values (for example, both AssumeLocal and AssumeUniversal).

Examples

The following example uses the DateTime.TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTime) method to ensure that a string in a number of possible formats can be successfully parsed .

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] formats= {"M/d/yyyy h:mm:ss tt", "M/d/yyyy h:mm tt",
                         "MM/dd/yyyy hh:mm:ss", "M/d/yyyy h:mm:ss",
```

```
                          "M/d/yyyy hh:mm tt", "M/d/yyyy hh tt",
                          "M/d/yyyy h:mm", "M/d/yyyy h:mm",
                          "MM/dd/yyyy hh:mm", "M/dd/yyyy hh:mm"};
      string[] dateStrings = {"5/1/2009 6:32 PM", "05/01/2009 6:32:05
PM",
                              "5/1/2009 6:32:00", "05/01/2009 06:32",
                              "05/01/2009 06:32:00 PM", "05/01/2009
06:32:00"};
      DateTime dateValue;

      foreach (string dateString in dateStrings)
      {
         if (DateTime.TryParseExact(dateString, formats,
                                    new CultureInfo("en-US"),
                                    DateTimeStyles.None,
                                    out dateValue))
            Console.WriteLine("Converted '{0}' to {1}.", dateString,
dateValue);
         else
            Console.WriteLine("Unable to convert '{0}' to a date.",
dateString);
      }
   }
}
// The example displays the following output:
//       Converted '5/1/2009 6:32 PM' to 5/1/2009 6:32:00 PM.
//       Converted '05/01/2009 6:32:05 PM' to 5/1/2009 6:32:05 PM.
//       Converted '5/1/2009 6:32:00' to 5/1/2009 6:32:00 AM.
//       Converted '05/01/2009 06:32' to 5/1/2009 6:32:00 AM.
//       Converted '05/01/2009 06:32:00 PM' to 5/1/2009 6:32:00 PM.
//       Converted '05/01/2009 06:32:00' to 5/1/2009 6:32:00 AM.
```

Remarks

The DateTime.TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime) method parses the string representation of a date that matches any one of the patterns assigned to the `formats` parameter. It is like the DateTime.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method, except the TryParseExact method does not throw an exception if the conversion fails.

The `s` parameter contains the date and time to parse. If the `s` parameter contains only a time and no date, the current date is used unless the `style` parameter includes the DateTimeStyles.NoCurrentDateDefault flag, in which case the default date (`DateTime.Date.MinValue`) is used. If the `s` parameter contains only a date and no time, midnight (00:00:00) is used. The `style` parameter also determines whether the `s` parameter can contain leading, inner, or trailing white space characters other than those permitted by one of the format strings in `formats`.

If `s` contains no time zone information, the Kind property of the returned DateTime object is DateTimeKind.Unspecified. This behavior can be changed by using the DateTimeStyles.AssumeLocal flag, which returns a DateTime value whose Kind property is DateTimeKind.Local, or by using the DateTimeStyles.AssumeUniversal and DateTimeStyles.AdjustToUniversal flags, which returns a DateTime value whose Kind property is DateTimeKind.Utc. If s contains time zone information, the time is converted to local time, if necessary, and the Kind property of the returned DateTime object is set to DateTimeKind.Local. This behavior can be changed by using the DateTimeStyles.RoundtripKind flag to not convert Coordinated Universal Time (UTC) to a local time and set the Kind property to DateTimeKind.Utc.

The `formats` parameter contains an array of patterns, one of which `s` must match exactly if the parse operation is to succeed. The patterns in the `formats` parameter consist of one or more custom format specifiers from the Custom Date and Time Format Strings table, or a single standard format specifier, which identifies a predefined pattern, from the Standard Date and Time Format Strings table.

If you do not use date or time separators in a custom format pattern, use the invariant culture for the `provider` parameter and the widest form of each custom format specifier. For example, if you want to specify hours in the pattern, specify the wider form, "HH", instead of the narrower form, "H".

The particular date and time symbols and strings (such as the names of the days of the week in a particular language) used in `s` are defined by the `provider` parameter, as is the precise format of `s` if `format` is a standard format specifier string. The `provider` parameter can be any of the following:

- A CultureInfo object that represents the culture used to interpret `s`. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `s`.
- A DateTimeFormatInfo object that defines the format of date and time data.
- A custom IFormatProvider implementation whose GetFormat method returns either the CultureInfo object or the DateTimeFormatInfo object that provides formatting information.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter includes one or more members of the DateTimeStyles enumeration that determine whether and where white space not defined by `format`

can appear in `s` and that control the precise behavior of the parse operation. The following table describes how each member of the DateTimeStyles enumeration affects the operation of the TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTime) method.

Expand table

DateTimeStyles member	Description
AdjustToUniversal	Parses `s` and, if necessary, converts it to UTC. If `s` includes a time zone offset, or if `s` contains no time zone information but `styles` includes the DateTimeStyles.AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc. If `s` indicates that it represents UTC, or if `s` does not contain time zone information but `styles` includes the DateTimeStyles.AssumeUniversal flag, the method parses the string, performs no time zone conversion on the returned DateTime value, and sets the Kind property to DateTimeKind.Utc. In all other cases, the flag has no effect.
AllowInnerWhite	Specifies that white space not defined by `format` can appear between any individual date or time element.
AllowLeadingWhite	Specifies that white space not defined by `format` can appear at the beginning of `s`.
AllowTrailingWhite	Specifies that white space not defined by `format` can appear at the end of `s`.
AllowWhiteSpaces	Specifies that `s` may contain leading, inner, and trailing white spaces not defined by `format`.
AssumeLocal	Specifies that if `s` lacks any time zone information, it is assumed to represent a local time. Unless the DateTimeStyles.AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local.
AssumeUniversal	Specifies that if `s` lacks any time zone information, it is assumed to represent UTC. Unless the DateTimeStyles.AdjustToUniversal flag is present, the method converts the returned DateTime value from UTC to local time and sets its Kind property to DateTimeKind.Local.
NoCurrentDateDefault	If `s` contains time without date information, the date of the return value is set to `DateTime.MinValue.Date`.
None	The `s` parameter is parsed using default values. No white space other than that present in `format` is allowed. If `s` lacks a date

DateTimeStyles member	Description
	component, the date of the returned DateTime value is set to 1/1/0001. If `s` contains no time zone information, the Kind property of the returned DateTime object is set to DateTimeKind.Unspecified. If time zone information is present in `s`, the time is converted to local time and the Kind property of the returned DateTime object is set to DateTimeKind.Local.
RoundtripKind	For strings that contain time zone information, tries to prevent the conversion to a DateTime value with its Kind property set to DateTimeKind.Local. This flag primarily prevents the conversion of UTC times to local times.

Notes to Callers

In the .NET Framework 4, the TryParseExact method returns `false` if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- ParseExact
- CultureInfo
- DateTimeFormatInfo
- Parsing Date and Time Strings in the .NET Framework
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Addition(DateTime, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds a specified time interval to a specified date and time, yielding a new date and time.

```
C#

public static DateTime operator + (DateTime d, TimeSpan t);
```

Parameters

`d` DateTime

The date and time value to add.

`t` TimeSpan

The time interval to add.

Returns

DateTime

An object that is the sum of the values of `d` and `t`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates the addition operator.

C#

```
System.DateTime dTime = new System.DateTime(1980, 8, 5);

// tSpan is 17 days, 4 hours, 2 minutes and 1 second.
System.TimeSpan tSpan
    = new System.TimeSpan(17, 4, 2, 1);

// Result gets 8/22/1980 4:02:01 AM.
System.DateTime result = dTime + tSpan;
```

Remarks

The equivalent method for this operator is DateTime.Add(TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(TimeSpan)
- TimeSpan
- Ticks
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTime.Equality(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of DateTime are equal.

```
C#

public static bool operator == (DateTime d1, DateTime d2);
```

Parameters

d1 DateTime

The first object to compare.

d2 DateTime

The second object to compare.

Returns

Boolean

`true` if `d1` and `d2` represent the same date and time; otherwise, `false`.

Examples

The following example demonstrates the equality operator.

```
C#

System.DateTime april19 = new DateTime(2001, 4, 19);
System.DateTime otherDate = new DateTime(1991, 6, 5);

// areEqual gets false.
bool areEqual = april19 == otherDate;
```

```
otherDate = new DateTime(2001, 4, 19);
// areEqual gets true.
areEqual = april19 == otherDate;
```

Remarks

The Equality operator determines whether two DateTime values are equal by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.GreaterThan(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTime is later than another specified DateTime.

```
C#
public static bool operator > (DateTime t1, DateTime t2);
```

Parameters

t1 DateTime

The first object to compare.

t2 DateTime

The second object to compare.

Returns

Boolean

`true` if `t1` is later than `t2`; otherwise, `false`.

Remarks

The GreaterThan operator determines the relationship between two DateTime values by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Compare(DateTime, DateTime)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.GreaterThanOrEqual(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTime represents a date and time that is the same as or later than another specified DateTime.

```
C#
public static bool operator >= (DateTime t1, DateTime t2);
```

Parameters

`t1` DateTime

The first object to compare.

`t2` DateTime

The second object to compare.

Returns

Boolean

`true` if `t1` is the same as or later than `t2`; otherwise, `false`.

Remarks

The GreaterThanOrEqual operator determines the relationship between two DateTime values by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Compare(DateTime, DateTime)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Inequality(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of DateTime are not equal.

```C#
public static bool operator != (DateTime d1, DateTime d2);
```

Parameters

d1 DateTime

The first object to compare.

d2 DateTime

The second object to compare.

Returns

Boolean

`true` if `d1` and `d2` do not represent the same date and time; otherwise, `false`.

Remarks

The Inequality operator determines whether two DateTime values are not equal by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.LessThan(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTime is earlier than another specified DateTime.

```
C#
public static bool operator < (DateTime t1, DateTime t2);
```

Parameters

t1 DateTime

The first object to compare.

t2 DateTime

The second object to compare.

Returns

Boolean

`true` if `t1` is earlier than `t2`; otherwise, `false`.

Remarks

The LessThan operator determines the relationship between two DateTime values by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Compare(DateTime, DateTime)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.LessThanOrEqual(DateTime, DateTime) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTime represents a date and time that is the same as or earlier than another specified DateTime.

C#

```
public static bool operator <= (DateTime t1, DateTime t2);
```

Parameters

`t1` DateTime

The first object to compare.

`t2` DateTime

The second object to compare.

Returns

Boolean

`true` if `t1` is the same as or earlier than `t2`; otherwise, `false`.

Remarks

The LessThanOrEqual operator determines the relationship between two DateTime values by comparing their number of ticks. Before comparing DateTime objects, make sure that the objects represent times in the same time zone. You can do this by comparing the values of their Kind property.

The equivalent method for this operator is DateTime.Compare(DateTime, DateTime)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts a specified DateTime orTimeSpan from a specified DateTime.

Overloads

Expand table

Subtraction(DateTime, DateTime)	Subtracts a specified date and time from another specified date and time and returns a time interval.
Subtraction(DateTime, TimeSpan)	Subtracts a specified time interval from a specified date and time and returns a new date and time.

Subtraction(DateTime, DateTime)

Subtracts a specified date and time from another specified date and time and returns a time interval.

C#

```
public static TimeSpan operator - (DateTime d1, DateTime d2);
```

Parameters

d1 DateTime

The date and time value to subtract from (the minuend).

d2 DateTime

The date and time value to subtract (the subtrahend).

Returns

TimeSpan

The time interval between `d1` and `d2`; that is, `d1` minus `d2`.

Examples

The following example demonstrates the Subtract method and the subtraction operator.

C#

```
System.DateTime date1 = new System.DateTime(1996, 6, 3, 22, 15, 0);
System.DateTime date2 = new System.DateTime(1996, 12, 6, 13, 2, 0);
System.DateTime date3 = new System.DateTime(1996, 10, 12, 8, 42, 0);

// diff1 gets 185 days, 14 hours, and 47 minutes.
System.TimeSpan diff1 = date2.Subtract(date1);

// date4 gets 4/9/1996 5:55:00 PM.
System.DateTime date4 = date3.Subtract(diff1);

// diff2 gets 55 days 4 hours and 20 minutes.
System.TimeSpan diff2 = date2 - date3;

// date5 gets 4/9/1996 5:55:00 PM.
System.DateTime date5 = date1 - diff2;
```

Remarks

The Subtraction(DateTime, DateTime) method does not consider the value of the Kind property of the two DateTime values when performing the subtraction. Before subtracting DateTime objects, ensure that the objects represent times in the same time zone. Otherwise, the result will include the difference between time zones.

> **Note**
>
> The **DateTimeOffset.Subtraction(DateTimeOffset, DateTimeOffset)** method does consider the difference between time zones when performing the subtraction.

The equivalent method for this operator is DateTime.Subtract(DateTime)

See also

- Subtract(DateTime)
- Ticks
- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Subtraction(DateTime, TimeSpan)

Subtracts a specified time interval from a specified date and time and returns a new date and time.

C#

```
public static DateTime operator - (DateTime d, TimeSpan t);
```

Parameters

`d` DateTime

The date and time value to subtract from.

`t` TimeSpan

The time interval to subtract.

Returns

DateTime

An object whose value is the value of `d` minus the value of `t`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTime is less than DateTime.MinValue or greater than DateTime.MaxValue.

Examples

The following example demonstrates the Subtract method and the subtraction operator.

C#

```
System.DateTime date1 = new System.DateTime(1996, 6, 3, 22, 15, 0);
System.DateTime date2 = new System.DateTime(1996, 12, 6, 13, 2, 0);
System.DateTime date3 = new System.DateTime(1996, 10, 12, 8, 42, 0);

// diff1 gets 185 days, 14 hours, and 47 minutes.
System.TimeSpan diff1 = date2.Subtract(date1);

// date4 gets 4/9/1996 5:55:00 PM.
System.DateTime date4 = date3.Subtract(diff1);

// diff2 gets 55 days 4 hours and 20 minutes.
System.TimeSpan diff2 = date2 - date3;

// date5 gets 4/9/1996 5:55:00 PM.
System.DateTime date5 = date1 - diff2;
```

Remarks

This method subtracts the ticks value of `t` from the ticks value of `d`.

The equivalent method for this operator is DateTime.Subtract(DateTime)

See also

- Subtract(TimeSpan)
- Ticks
- TimeSpan
- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Boolean

The return value for this member is not used.

Implements

ToBoolean(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Byte

The return value for this member is not used.

Implements

ToByte(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Char

The return value for this member is not used.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the current DateTime object.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

DateTime

The current object.

Implements

ToDateTime(IFormatProvider)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Decimal

The return value for this member is not used.

Implements

ToDecimal(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Double

The return value for this member is not used.

Implements

ToDouble(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int16

The return value for this member is not used.

Implements

ToInt16(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int32

The return value for this member is not used.

Implements

ToInt32(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int64

The return value for this member is not used.

Implements

ToInt64(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

SByte

The return value for this member is not used.

Implements

ToSByte(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Single

The return value for this member is not used.

Implements

ToSingle(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current DateTime object to an object of a specified type.

```
C#
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The desired type.

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Object

An object of the type specified by the `type` parameter, with a value equivalent to the current DateTime object.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

This conversion is not supported for the DateTime type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Convert

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt16

The return value for this member is not used.

Implements

ToUInt16(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt32

The return value for this member is not used.

Implements

ToUInt32(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt64

The return value for this member is not used.

Implements

ToUInt64(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTime.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data needed to serialize the current DateTime object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with data.

`context` StreamingContext

The destination for this serialization. (This parameter is not used; specify `null`.)

Implements

GetObjectData(SerializationInfo, StreamingContext)

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

This method populates the `info` parameter with the Ticks and Kind property values.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeKind Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies whether a DateTime object represents a local time, a Coordinated Universal Time (UTC), or is not specified as either local time or UTC.

```
C#

public enum DateTimeKind
```

Inheritance Object → ValueType → Enum → DateTimeKind

Fields

Expand table

Local	2	The time represented is local time.
Unspecified	0	The time represented is not specified as either local time or Coordinated Universal Time (UTC).
Utc	1	The time represented is UTC.

Remarks

A member of the DateTimeKind enumeration is returned by the DateTime.Kind property.

The members of the DateTimeKind enumeration are used in conversion operations between local time and Coordinated Universal Time (UTC), but not in comparison or arithmetic operations. For more information about time conversions, see Converting Times Between Time Zones.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a point in time, typically expressed as a date and time of day, relative to Coordinated Universal Time (UTC).

C#

```
public readonly struct DateTimeOffset : IComparable,
IComparable<DateTimeOffset>, IEquatable<DateTimeOffset>,
IParsable<DateTimeOffset>, ISpanFormattable, ISpanParsable<DateTimeOffset>,
IUtf8SpanFormattable, System.Runtime.Serialization.IDeserializationCallback,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → DateTimeOffset

Implements IComparable , IComparable<DateTimeOffset> , IEquatable<DateTimeOffset> , IFormattable , IDeserializationCallback , ISerializable , ISpanFormattable , IParsable<DateTimeOffset> , IParsable<TSelf> , ISpanParsable<DateTimeOffset> , IUtf8SpanFormattable

Remarks

The DateTimeOffset structure includes a DateTime value, together with an Offset property that defines the difference between the current DateTimeOffset instance's date and time and Coordinated Universal Time (UTC). Because it exactly defines a date and time relative to UTC, the DateTimeOffset structure does not include a `Kind` member, as the DateTime structure does. It represents dates and times with values whose UTC ranges from 12:00:00 midnight, January 1, 0001 Anno Domini (Common Era), to 11:59:59 P.M., December 31, 9999 A.D. (C.E.).

Important

Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

The time component of a DateTimeOffset value is measured in 100-nanosecond units called ticks, and a particular date is the number of ticks since 12:00 midnight, January 1, 0001 A.D. (C.E.) in the GregorianCalendar calendar. A DateTimeOffset value is always expressed in the context of an explicit or default calendar. Ticks that are attributable to leap seconds are not included in the total number of ticks.

Although a DateTimeOffset value includes an offset, it is not a fully time zone-aware data structure. While an offset from UTC is one characteristic of a time zone, it does not unambiguously identify a time zone. Not only do multiple time zones share the same offset from UTC, but the offset of a single time zone changes if it observes daylight saving time. This means that, as soon as a DateTimeOffset value is disassociated from its time zone, it can no longer be unambiguously linked back to its original time zone.

Because DateTimeOffset is a structure, a DateTimeOffset object that has been declared but not otherwise initialized contains the default values for each of its member fields. This means that its DateTime property is set to DateTimeOffset.MinValue and its Offset property is set to TimeSpan.Zero.

You can create a new DateTimeOffset value by calling any of the overloads of its constructor, which are similar to the overloaded constructors for the DateTime structure. You can also create a new DateTimeOffset value by assigning it a DateTime value. This is an implicit conversion; it does not require a casting operator (in C#) or call to a conversion method (in Visual Basic). You can also initialize a DateTimeOffset value from the string representation of a date and time by calling a number of static string parsing methods, which include Parse, ParseExact, TryParse, and TryParseExact.

The members of the DateTimeOffset structure provide functionality in the following areas:

- Date and time arithmetic.

You can add or subtract either dates or time intervals from a particular DateTimeOffset value. Arithmetic operations with DateTimeOffset values, unlike those with DateTime values, adjust for differences in time offsets when returning a result. For example, the following code uses DateTime variables to subtract the current local time from the current UTC time. The code then uses DateTimeOffset variables to perform the same operation. The subtraction with DateTime values returns the local time zone's difference from UTC, while the subtraction with DateTimeOffset values returns TimeSpan.Zero.

C#

```
using System;

public class DateArithmetic
{
   public static void Main()
   {
      DateTime date1, date2;
      DateTimeOffset dateOffset1, dateOffset2;
      TimeSpan difference;

      // Find difference between Date.Now and Date.UtcNow
      date1 = DateTime.Now;
      date2 = DateTime.UtcNow;
      difference = date1 - date2;
      Console.WriteLine("{0} - {1} = {2}", date1, date2, difference);

      // Find difference between Now and UtcNow using DateTimeOffset
      dateOffset1 = DateTimeOffset.Now;
      dateOffset2 = DateTimeOffset.UtcNow;
      difference = dateOffset1 - dateOffset2;
      Console.WriteLine("{0} - {1} = {2}",
                        dateOffset1, dateOffset2, difference);
      // If run in the Pacific Standard time zone on 4/2/2007, the
example
      // displays the following output to the console:
      //    4/2/2007 7:23:57 PM - 4/3/2007 2:23:57 AM = -07:00:00
      //    4/2/2007 7:23:57 PM -07:00 - 4/3/2007 2:23:57 AM +00:00 =
00:00:00
   }
}
```

- Type conversion operations.

  You can convert DateTimeOffset values to DateTime values and vice versa.

- Time manipulation and extraction operations.

You can extract either the date or the time of a DateTimeOffset value. You can also retrieve the value of a particular DateTimeOffset component, such as its year or its month.

> **Note**
>
> If you are working with a ticks value that you want to convert to some other time interval, such as minutes or seconds, you should use the **TimeSpan.TicksPerDay**, **TimeSpan.TicksPerHour**, **TimeSpan.TicksPerMinute**, **TimeSpan.TicksPerSecond**, or **TimeSpan.TicksPerMillisecond** constant to perform the conversion. For example, to add the number of seconds represented by a specified number of ticks to the **Second** component of a **DateTimeOffset** value, you can use the expression `dateValue.Second + nTicks/Timespan.TicksPerSecond`.

- Date and time conversion.

  You can convert any DateTimeOffset value to another DateTimeOffset value that represents the same point in time in another time zone. However, a time zone's adjustment rules are applied only in the case of the ToLocalTime method, which converts a DateTimeOffset value to the date and time in the local system zone.

- Date and time comparison.

  You can determine whether any particular DateTimeOffset value is earlier than, the same as, or later than another DateTimeOffset value. Before the comparison is performed, all values are converted to UTC.

Constructors

Expand table

DateTimeOffset(DateOnly, TimeOnly, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified `date`, `time`, and `offset`.
DateTimeOffset(DateTime)	Initializes a new instance of the DateTimeOffset structure using the specified DateTime value.
DateTimeOffset(DateTime, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified DateTime value and offset.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset of a specified calendar.

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Time Span)	Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, and offset.
DateTimeOffset(Int64, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified number of ticks and offset.

Fields

Expand table

Max Value	Represents the greatest possible value of DateTimeOffset. This field is read-only.
Min Value	Represents the earliest possible DateTimeOffset value. This field is read-only.
Unix Epoch	The value of this constant is equivalent to 00:00:00.0000000 UTC, January 1, 1970, in the Gregorian calendar. UnixEpoch defines the point in time when Unix time is equal to 0.

Properties

Expand table

Date	Gets a DateTime value that represents the date component of the current DateTimeOffset object.
DateTime	Gets a DateTime value that represents the date and time of the current DateTimeOffset object.
Day	Gets the day of the month represented by the current DateTimeOffset object.
DayOfWeek	Gets the day of the week represented by the current DateTimeOffset object.

DayOfYear	Gets the day of the year represented by the current DateTimeOffset object.
Hour	Gets the hour component of the time represented by the current DateTimeOffset object.
LocalDateTime	Gets a DateTime value that represents the local date and time of the current DateTimeOffset object.
Microsecond	Gets the microsecond component of the time represented by the current DateTimeOffset object.
Millisecond	Gets the millisecond component of the time represented by the current DateTimeOffset object.
Minute	Gets the minute component of the time represented by the current DateTimeOffset object.
Month	Gets the month component of the date represented by the current DateTimeOffset object.
Nanosecond	Gets the nanosecond component of the time represented by the current DateTimeOffset object.
Now	Gets a DateTimeOffset object that is set to the current date and time on the current computer, with the offset set to the local time's offset from Coordinated Universal Time (UTC).
Offset	Gets the time's offset from Coordinated Universal Time (UTC).
Second	Gets the second component of the clock time represented by the current DateTimeOffset object.
Ticks	Gets the number of ticks that represents the date and time of the current DateTimeOffset object in clock time.
TimeOfDay	Gets the time of day for the current DateTimeOffset object.
TotalOffset Minutes	Gets the time's offset from Coordinated Universal Time (UTC) in minutes.
UtcDateTime	Gets a DateTime value that represents the Coordinated Universal Time (UTC) date and time of the current DateTimeOffset object.
UtcNow	Gets a DateTimeOffset object whose date and time are set to the current Coordinated Universal Time (UTC) date and time and whose offset is Zero.
UtcTicks	Gets the number of ticks that represents the date and time of the current DateTimeOffset object in Coordinated Universal Time (UTC).
Year	Gets the year component of the date represented by the current DateTimeOffset object.

Methods

Expand table

Add(TimeSpan)	Returns a new DateTimeOffset object that adds a specified time interval to the value of this instance.
AddDays(Double)	Returns a new DateTimeOffset object that adds a specified number of whole and fractional days to the value of this instance.
AddHours(Double)	Returns a new DateTimeOffset object that adds a specified number of whole and fractional hours to the value of this instance.
AddMicroseconds(Double)	Returns a new DateTimeOffset object that adds a specified number of microseconds to the value of this instance.
AddMilliseconds(Double)	Returns a new DateTimeOffset object that adds a specified number of milliseconds to the value of this instance.
AddMinutes(Double)	Returns a new DateTimeOffset object that adds a specified number of whole and fractional minutes to the value of this instance.
AddMonths(Int32)	Returns a new DateTimeOffset object that adds a specified number of months to the value of this instance.
AddSeconds(Double)	Returns a new DateTimeOffset object that adds a specified number of whole and fractional seconds to the value of this instance.
AddTicks(Int64)	Returns a new DateTimeOffset object that adds a specified number of ticks to the value of this instance.
AddYears(Int32)	Returns a new DateTimeOffset object that adds a specified number of years to the value of this instance.
Compare(DateTimeOffset, DateTime Offset)	Compares two DateTimeOffset objects and indicates whether the first is earlier than the second, equal to the second, or later than the second.
CompareTo(DateTimeOffset)	Compares the current DateTimeOffset object to a specified DateTimeOffset object and indicates whether the current object is earlier than, the same as, or later than the second DateTimeOffset object.

Deconstruct(DateOnly, TimeOnly, TimeSpan)	Deconstructs this DateTimeOffset instance by DateOnly, TimeOnly, and TimeSpan.
Equals(DateTimeOffset)	Determines whether the current DateTimeOffset object represents the same point in time as a specified DateTimeOffset object.
Equals(DateTimeOffset, DateTimeOffset)	Determines whether two specified DateTimeOffset objects represent the same point in time.
Equals(Object)	Determines whether a DateTimeOffset object represents the same point in time as a specified object.
EqualsExact(DateTimeOffset)	Determines whether the current DateTimeOffset object represents the same time and has the same offset as a specified DateTimeOffset object.
FromFileTime(Int64)	Converts the specified Windows file time to an equivalent local time.
FromUnixTimeMilliseconds(Int64)	Converts a Unix time expressed as the number of milliseconds that have elapsed since 1970-01-01T00:00:00Z to a DateTimeOffset value.
FromUnixTimeSeconds(Int64)	Converts a Unix time expressed as the number of seconds that have elapsed since 1970-01-01T00:00:00Z to a DateTimeOffset value.
GetHashCode()	Returns the hash code for the current DateTimeOffset object.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.
Parse(String)	Converts the specified string representation of a date, time, and offset to its DateTimeOffset equivalent.
Parse(String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a character span that represents a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts a character span that contains the string representation of a date and time to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.
ParseExact(String, String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.
ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.
Subtract(DateTimeOffset)	Subtracts a DateTimeOffset value that represents a specific date and time from the current DateTimeOffset object.
Subtract(TimeSpan)	Subtracts a specified time interval from the current DateTimeOffset object.
ToFileTime()	Converts the value of the current DateTimeOffset object to a Windows file time.
ToLocalTime()	Converts the current DateTimeOffset object to a DateTimeOffset object that represents the local time.
ToOffset(TimeSpan)	Converts the value of the current DateTimeOffset object to the date and time specified by an offset value.
ToString()	Converts the value of the current DateTimeOffset object to its equivalent string representation.

ToString(IFormatProvider)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified culture-specific formatting information.
ToString(String)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format and culture-specific format information.
ToUniversalTime()	Converts the current DateTimeOffset object to a DateTimeOffset value that represents the Coordinated Universal Time (UTC).
ToUnixTimeMilliseconds()	Returns the number of milliseconds that have elapsed since 1970-01-01T00:00:00.000Z.
ToUnixTimeSeconds()	Returns the number of seconds that have elapsed since 1970-01-01T00:00:00Z.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current datetime offset instance into the provided span of characters.
TryParse(ReadOnlySpan<Char>, DateTimeOffset)	Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeOffset)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)	Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, DateTimeOffset)	Tries to converts a specified string representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateTimeOffset)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset)	Tries to convert a specified string representation of a date and time to its DateTimeOffset equivalent, and

	returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.

Operators

Expand table

Addition(DateTimeOffset, TimeSpan)	Adds a specified time interval to a DateTimeOffset object that has a specified date and time, and yields a DateTimeOffset object that has new a date and time.
Equality(DateTimeOffset, DateTimeOffset)	Determines whether two specified DateTimeOffset objects represent the same point in time.
GreaterThan(DateTimeOffset, DateTimeOffset)	Determines whether one specified DateTimeOffset object is greater than (or later than) a second specified DateTimeOffset object.
GreaterThanOrEqual(DateTimeOffset, DateTimeOffset)	Determines whether one specified DateTimeOffset object is greater than or equal to a second specified DateTimeOffset object.

Implicit(DateTime to DateTimeOffset)	Defines an implicit conversion of a DateTime object to a DateTimeOffset object.
Inequality(DateTimeOffset, DateTimeOffset)	Determines whether two specified DateTimeOffset objects refer to different points in time.
LessThan(DateTimeOffset, DateTimeOffset)	Determines whether one specified DateTimeOffset object is less than a second specified DateTimeOffset object.
LessThanOrEqual(DateTimeOffset, DateTimeOffset)	Determines whether one specified DateTimeOffset object is less than a second specified DateTimeOffset object.
Subtraction(DateTimeOffset, DateTimeOffset)	Subtracts one DateTimeOffset object from another and yields a time interval.
Subtraction(DateTimeOffset, TimeSpan)	Subtracts a specified time interval from a specified date and time, and yields a new date and time.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the value of the current DateTimeOffset object with another object of the same type.
IDeserializationCallback.OnDeserialization(Object)	Runs when the deserialization of an object has been completed.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data required to serialize the current DateTimeOffset object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Choosing Between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the DateTimeOffset structure.

Overloads

Expand table

DateTimeOffset(DateTime)	Initializes a new instance of the DateTimeOffset structure using the specified DateTime value.
DateTimeOffset(DateTime, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified DateTime value and offset.
DateTimeOffset(Int64, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified number of ticks and offset.
DateTimeOffset(DateOnly, TimeOnly, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified `date`, `time`, and `offset`.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, and offset.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset of a specified calendar.
DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)	Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.

DateTimeOffset(DateTime)

Initializes a new instance of the DateTimeOffset structure using the specified DateTime value.

```
C#

public DateTimeOffset (DateTime dateTime);
```

Parameters

`dateTime` DateTime

A date and time.

Exceptions

ArgumentOutOfRangeException

The Coordinated Universal Time (UTC) date and time that results from applying the offset is earlier than DateTimeOffset.MinValue.

-or-

The UTC date and time that results from applying the offset is later than DateTimeOffset.MaxValue.

Examples

The following example illustrates how the value of the DateTime.Kind property of the `dateTime` parameter affects the date and time value that is returned by this constructor.

```
C#

DateTime localNow = DateTime.Now;
DateTimeOffset localOffset = new DateTimeOffset(localNow);
Console.WriteLine(localOffset.ToString());

DateTime utcNow = DateTime.UtcNow;
```

```
DateTimeOffset utcOffset = new DateTimeOffset(utcNow);
Console.WriteLine(utcOffset.ToString());

DateTime unspecifiedNow = DateTime.SpecifyKind(DateTime.Now,
                                               DateTimeKind.Unspecified);
DateTimeOffset unspecifiedOffset = new DateTimeOffset(unspecifiedNow);
Console.WriteLine(unspecifiedOffset.ToString());
//
// The code produces the following output if run on Feb. 23, 2007, on
// a system 8 hours earlier than UTC:
//   2/23/2007 4:21:58 PM -08:00
//   2/24/2007 12:21:58 AM +00:00
//   2/23/2007 4:21:58 PM -08:00
```

Remarks

This constructor's behavior depends on the value of the DateTime.Kind property of the `dateTime` parameter:

- If the value of DateTime.Kind is DateTimeKind.Utc, the DateTime property of the new instance is set equal to `dateTime`, and the Offset property is set equal to Zero.
- If the value of DateTime.Kind is DateTimeKind.Local or DateTimeKind.Unspecified, the DateTime property of the new instance is set equal to `dateTime`, and the Offset property is set equal to the offset of the local system's current time zone.

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

DateTimeOffset(DateTime, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified DateTime value and offset.

```
C#
public DateTimeOffset (DateTime dateTime, TimeSpan offset);
```

Parameters

`dateTime` DateTime

A date and time.

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`dateTime.Kind` equals Utc and `offset` does not equal zero.

-or-

`dateTime.Kind` equals Local and `offset` does not equal the offset of the system's local time zone.

-or-

`offset` is not specified in whole minutes.

ArgumentOutOfRangeException

`offset` is less than -14 hours or greater than 14 hours.

-or-

UtcDateTime is less than DateTimeOffset.MinValue or greater than DateTimeOffset.MaxValue.

Examples

The following example shows how to initialize a DateTimeOffset object with a date and time and the offset of the local time zone when that time zone is not known in advance.

```C#
DateTime localTime = new DateTime(2007, 07, 12, 06, 32, 00);
DateTimeOffset dateAndOffset = new DateTimeOffset(localTime,
                          TimeZoneInfo.Local.GetUtcOffset(localTime));
Console.WriteLine(dateAndOffset);
// The code produces the following output:
//    7/12/2007 6:32:00 AM -07:00
```

Remarks

This constructor's behavior depends in part on the value of the Kind property of the `dateTime` parameter:

- If the value of Kind is DateTimeKind.Utc, the value of the `offset` parameter must be 0 or an ArgumentException is thrown.
- If the value of Kind is DateTimeKind.Local, the value of the `offset` parameter must be equal to the local time zone's offset from Coordinated Universal Time (UTC) for that particular date or an ArgumentException is thrown.
- If the value of Kind is DateTimeKind.Unspecified, the `offset` parameter can have any valid value.

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTimeOffset(Int64, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified number of ticks and offset.

```
C#

public DateTimeOffset (long ticks, TimeSpan offset);
```

Parameters

`ticks` Int64

A date and time expressed as the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight on January 1, 0001.

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` is not specified in whole minutes.

ArgumentOutOfRangeException

The UtcDateTime property is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue.

-or-

`ticks` is less than `DateTimeOffset.MinValue.Ticks` or greater than `DateTimeOffset.MaxValue.Ticks`.

-or-

`Offset` s less than -14 hours or greater than 14 hours.

Examples

The following example initializes a DateTimeOffset object by using the number of ticks in an arbitrary date (in this case, July 16, 2007, at 1:32 PM) with an offset of -5.

C#

```
DateTime dateWithoutOffset = new DateTime(2007, 7, 16, 13, 32, 00);
DateTimeOffset timeFromTicks = new
DateTimeOffset(dateWithoutOffset.Ticks,
                                new TimeSpan(-5, 0, 0));
Console.WriteLine(timeFromTicks.ToString());
// The code produces the following output:
//    7/16/2007 1:32:00 PM -05:00
```

Remarks

Ordinarily, trying to call the DateTimeOffset constructor to instantiate a DateTimeOffset value with a local time and an offset other than that of the local time zone throws an ArgumentException. You can use this overload of the DateTimeOffset constructor to work around this limitation. The following example uses the local time's number of ticks to instantiate a DateTimeOffset value whose offset does not necessarily represent that of the local time:

C#

```
DateTime localTime = DateTime.Now;
DateTimeOffset nonLocalDateWithOffset = new
DateTimeOffset(localTime.Ticks,
                                  new TimeSpan(2, 0, 0));
Console.WriteLine(nonLocalDateWithOffset);
//
// The code produces the following output if run on Feb. 23, 2007:
//    2/23/2007 4:37:50 PM +02:00
```

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTimeOffset(DateOnly, TimeOnly, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified `date`, `time`, and `offset`.

```
C#

public DateTimeOffset (DateOnly date, TimeOnly time, TimeSpan offset);
```

Parameters

`date` DateOnly

The date part.

`time` TimeOnly

The time part.

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, and offset.

C#

```
public DateTimeOffset (int year, int month, int day, int hour, int
minute, int second, TimeSpan offset);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` does not represent whole minutes.

ArgumentOutOfRangeException

`year` is less than one or greater than 9999.

-or-

`month` is less than one or greater than 12.

-or-

`day` is less than one or greater than the number of days in `month`.

-or-

`hour` is less than zero or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`offset` is less than -14 hours or greater than 14 hours.

-or-

The UtcDateTime property is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue.

Examples

The following example instantiates a DateTimeOffset object by using the DateTimeOffset.DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan) constructor overload.

C#

```
DateTime specificDate = new DateTime(2008, 5, 1, 06, 32, 00);
DateTimeOffset offsetDate = new DateTimeOffset(specificDate.Year,
                                specificDate.Month,
                                specificDate.Day,
                                specificDate.Hour,
                                specificDate.Minute,
                                specificDate.Second,
                                new TimeSpan(-5, 0, 0));
Console.WriteLine("Current time: {0}", offsetDate);
```

```
    Console.WriteLine("Corresponding UTC time: {0}",
offsetDate.UtcDateTime);
// The code produces the following output:
//    Current time: 5/1/2008 6:32:00 AM -05:00
//    Corresponding UTC time: 5/1/2008 11:32:00 AM
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTimeOffset value by using the year, month, and day in another calendar, call the DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan) constructor.

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset.

C#

```
public DateTimeOffset (int year, int month, int day, int hour, int
```

```
minute, int second, int millisecond, TimeSpan offset);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` does not represent whole minutes.

ArgumentOutOfRangeException

`year` is less than one or greater than 9999.

-or-

`month` is less than one or greater than 12.

-or-

`day` is less than one or greater than the number of days in `month`.

-or-

`hour` is less than zero or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`offset` is less than -14 or greater than 14.

-or-

The UtcDateTime property is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue.

Examples

The following example instantiates a DateTimeOffset object by using the DateTimeOffset.DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan) constructor overload.

C#

```
string fmt = "dd MMM yyyy HH:mm:ss";
DateTime thisDate = new DateTime(2007, 06, 12, 19, 00, 14, 16);
DateTimeOffset offsetDate = new DateTimeOffset(thisDate.Year,
                                               thisDate.Month,
                                               thisDate.Day,
                                               thisDate.Hour,
                                               thisDate.Minute,
                                               thisDate.Second,
                                               thisDate.Millisecond,
                                               new TimeSpan(2, 0, 0));
Console.WriteLine("Current time: {0}:{1}", offsetDate.ToString(fmt),
offsetDate.Millisecond);
```

```
// The code produces the following output:
//    Current time: 12 Jun 2007 19:00:14:16
```

Remarks

This constructor interprets `year`, `month`, and `day` as a year, month, and day in the Gregorian calendar. To instantiate a DateTimeOffset value by using the year, month, and day in another calendar, call the DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan) constructor.

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified year, month, day, hour, minute, second, millisecond, and offset of a specified calendar.

C#

```
public DateTimeOffset (int year, int month, int day, int hour, int
minute, int second, int millisecond, System.Globalization.Calendar
calendar, TimeSpan offset);
```

Parameters

`year` Int32

The year.

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` does not represent whole minutes.

ArgumentNullException

`calendar` cannot be `null`.

ArgumentOutOfRangeException

`year` is less than the `calendar` parameter's `MinSupportedDateTime.Year` or greater than `MaxSupportedDateTime.Year`.

-or-

`month` is either less than or greater than the number of months in `year` in the `calendar`.

-or-

`day` is less than one or greater than the number of days in `month`.

-or-

`hour` is less than zero or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`offset` is less than -14 hours or greater than 14 hours.

-or-

The `year`, `month`, and `day` parameters cannot be represented as a date and time value.

-or-

The UtcDateTime property is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue.

Examples

The following example uses instances of both the HebrewCalendar class and the HijriCalendar class to instantiate a DateTimeOffset value. That date is then displayed to the console using the respective calendars and the Gregorian calendar.

```
C#
```

```
CultureInfo fmt;
int year;
Calendar cal;
DateTimeOffset dateInCal;

// Instantiate DateTimeOffset with Hebrew calendar
year = 5770;
cal = new HebrewCalendar();
fmt = new CultureInfo("he-IL");
fmt.DateTimeFormat.Calendar = cal;
dateInCal = new DateTimeOffset(year, 7, 12,
                               15, 30, 0, 0,
                               cal,
                               new TimeSpan(2, 0, 0));
// Display the date in the Hebrew calendar
Console.WriteLine("Date in Hebrew Calendar: {0:g}",
                  dateInCal.ToString(fmt));
// Display the date in the Gregorian calendar
Console.WriteLine("Date in Gregorian Calendar: {0:g}", dateInCal);
Console.WriteLine();

// Instantiate DateTimeOffset with Hijri calendar
year = 1431;
cal = new HijriCalendar();
fmt = new CultureInfo("ar-SA");
fmt.DateTimeFormat.Calendar = cal;
dateInCal = new DateTimeOffset(year, 7, 12,
                               15, 30, 0, 0,
                               cal,
                               new TimeSpan(2, 0, 0));
// Display the date in the Hijri calendar
Console.WriteLine("Date in Hijri Calendar: {0:g}",
                  dateInCal.ToString(fmt));
// Display the date in the Gregorian calendar
Console.WriteLine("Date in Gregorian Calendar: {0:g}", dateInCal);
Console.WriteLine();
```

Remarks

The `year`, `month`, `day`, `hour`, `minute`, `second`, and `millisecond` parameters all reflect values expressed in the calendar specified by the `calendar` parameter. An exception is thrown if these values form a date and time that cannot be expressed by using this calendar.

> **ⓘ Important**
>
> Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning

of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

See also

- Instantiating a DateTimeOffset Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.

C#

```
public DateTimeOffset (int year, int month, int day, int hour, int
minute, int second, int millisecond, int microsecond, TimeSpan offset);
```

Parameters

year Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` does not represent whole minutes.

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

-or-

`month` is less than 1 or greater than 12.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

Remarks

This constructor interprets `year`, `month` and `day` as a year, month and day in the Gregorian calendar. To instantiate a DateTimeOffset value by using the year, month and day in another calendar, call the DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan) constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan)

Initializes a new instance of the DateTimeOffset structure using the specified `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond` and `offset`.

C#

```
public DateTimeOffset (int year, int month, int day, int hour, int
minute, int second, int millisecond, int microsecond,
```

```
System.Globalization.Calendar calendar, TimeSpan offset);
```

Parameters

`year` Int32

The year (1 through 9999).

`month` Int32

The month (1 through 12).

`day` Int32

The day (1 through the number of days in `month`).

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The milliseconds (0 through 999).

`microsecond` Int32

The microseconds (0 through 999).

`calendar` Calendar

The calendar that is used to interpret `year`, `month`, and `day`.

`offset` TimeSpan

The time's offset from Coordinated Universal Time (UTC).

Exceptions

ArgumentException

`offset` does not represent whole minutes.

ArgumentOutOfRangeException

`year` is not in the range supported by `calendar`.

-or-

`month` is less than 1 or greater than the number of months in `calendar`.

-or-

`day` is less than 1 or greater than the number of days in `month`.

-or-

`hour` is less than 0 or greater than 23.

-or-

`minute` is less than 0 or greater than 59.

-or-

`second` is less than 0 or greater than 59.

-or-

`millisecond` is less than 0 or greater than 999.

-or-

`microsecond` is less than 0 or greater than 999.

-or-

`offset` is less than -14 hours or greater than 14 hours.

-or-

The `year`, `month`, and `day` parameters cannot be represented as a date and time value.

-or-

The UtcDateTime property is earlier than MinValue or later than MaxValue.

Remarks

This constructor interprets `year`, `month` and `day` as a year, month and day in the Gregorian calendar. To instantiate a DateTimeOffset value by using the year, month and day in another calendar, call the DateTimeOffset(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, TimeSpan) constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the greatest possible value of DateTimeOffset. This field is read-only.

```
C#
public static readonly DateTimeOffset MaxValue;
```

Field Value

DateTimeOffset

Exceptions

ArgumentOutOfRangeException

DateTime.MaxValue is outside the range of the current or specified culture's default calendar.

Remarks

The value of this constant is 12/31/9999 11:59:59 PM +00:00.

The MaxValue property is used to determine whether the value of a new DateTimeOffset object or the DateTimeOffset value returned by an arithmetic operation is the same as or earlier than this maximum range value. If it is not, the method throws an ArgumentOutOfRangeException. Any DateTimeOffset value is converted to Coordinated Universal Time (UTC) before the method compares it with MaxValue. This means that a DateTimeOffset value whose date and time are close to the maximum range, but whose offset is negative, may throw an exception. For example, the value 12/31/9999 11:00 PM -02:00 is out of range because it is one hour later than MaxValue when it is converted to UTC.

Some calendars, such as the UmAlQuraCalendar, support an upper date range that is earlier than MaxValue. In these cases, trying to access MaxValue in variable assignments

or formatting and parsing operations can throw an ArgumentOutOfRangeException. Instead of retrieving the value of DateTime.MaxValue, you can retrieve the value of the specified culture's latest valid date value from the Calendar.MaxSupportedDateTime property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the earliest possible DateTimeOffset value. This field is read-only.

```
C#
public static readonly DateTimeOffset MinValue;
```

Field Value

DateTimeOffset

Remarks

The value of this constant is 1/1/0001 12:00:00 AM +00:00.

The MinValue property is used to determine whether the value of a new DateTimeOffset object or the DateTimeOffset value returned by an arithmetic operation is the same as or later than this minimum range value. If it is not, the method throws an ArgumentOutOfRangeException. Any DateTimeOffset value is converted to Coordinated Universal Time (UTC) before the method performs the comparison with MinValue. This means that a DateTimeOffset value whose date and time are close to the minimum range, but whose offset is positive, may throw an exception. For example, the value 1/1/0001 1:00:00 AM +02:00 is out of range because it is one hour earlier than MinValue when it is converted to UTC.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ParseExact

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.UnixEpoch Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The value of this constant is equivalent to 00:00:00.0000000 UTC, January 1, 1970, in the Gregorian calendar. UnixEpoch defines the point in time when Unix time is equal to 0.

C#

```
public static readonly DateTimeOffset UnixEpoch;
```

Field Value

DateTimeOffset

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Date Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime value that represents the date component of the current DateTimeOffset object.

C#

```
public DateTime Date { get; }
```

Property Value

DateTime

A DateTime value that represents the date component of the current DateTimeOffset object.

Examples

The following example retrieves the value of the Date property for a specific date. It then displays that value to the console using some standard and custom date-only format specifiers.

C#

```
// Illustrate Date property and date formatting
DateTimeOffset thisDate = new DateTimeOffset(2008, 3, 17, 1, 32, 0, new
TimeSpan(-5, 0, 0));
string fmt;                       // format specifier

// Display date only using "D" format specifier
// For en-us culture, displays:
//   'D' format specifier: Monday, March 17, 2008
fmt = "D";
Console.WriteLine("'{0}' format specifier: {1}",
                  fmt, thisDate.Date.ToString(fmt));

// Display date only using "d" format specifier
```

```
// For en-us culture, displays:
//   'd' format specifier: 3/17/2008
fmt = "d";
Console.WriteLine("'{0}' format specifier: {1}",
                  fmt, thisDate.Date.ToString(fmt));

// Display date only using "Y" (or "y") format specifier
// For en-us culture, displays:
//   'Y' format specifier: March, 2008
fmt = "Y";
Console.WriteLine("'{0}' format specifier: {1}",
                  fmt, thisDate.Date.ToString(fmt));

// Display date only using custom format specifier
// For en-us culture, displays:
//   'dd MMM yyyy' format specifier: 17 Mar 2008
fmt = "dd MMM yyyy";
Console.WriteLine("'{0}' format specifier: {1}",
                  fmt, thisDate.Date.ToString(fmt));
```

Remarks

This property removes any significant part of the time component from a DateTimeOffset object and returns only its significant date component. For example, if the DateTimeOffset object has a date and time value of "1/12/07 4:01pm +7:30", the property returns a DateTime value of "1/12/07 12:00:00 AM". The DateTime value can then be displayed by using any of the standard or custom format specifiers that display dates only. (See the Example section for an illustration.)

The value of the DateTime.Kind property of the returned DateTime object is always DateTimeKind.Unspecified. It is not affected by the value of the Offset property.

To display a date without its time component, you can also use the "D" or "d" format specifiers; for an illustration, see the Example section.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Standard DateTime Format Strings
- Custom DateTime Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.DateTime Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime value that represents the date and time of the current DateTimeOffset object.

C#

```
public DateTime DateTime { get; }
```

Property Value

DateTime

The date and time of the current DateTimeOffset object.

Examples

The following example illustrates the use of the DateTime property to convert the time returned by the Now and UtcNow properties to DateTime values.

C#

```
DateTimeOffset offsetDate;
DateTime regularDate;

offsetDate = DateTimeOffset.Now;
regularDate = offsetDate.DateTime;
Console.WriteLine("{0} converts to {1}, Kind {2}.",
                  offsetDate.ToString(),
                  regularDate,
                  regularDate.Kind);

offsetDate = DateTimeOffset.UtcNow;
regularDate = offsetDate.DateTime;
Console.WriteLine("{0} converts to {1}, Kind {2}.",
                  offsetDate.ToString(),
                  regularDate,
                  regularDate.Kind);
```

```
// If run on 3/6/2007 at 17:11, produces the following output:
//
//   3/6/2007 5:11:22 PM -08:00 converts to 3/6/2007 5:11:22 PM, Kind
Unspecified.
//   3/7/2007 1:11:22 AM +00:00 converts to 3/7/2007 1:11:22 AM, Kind
Unspecified.
```

Remarks

The DateTime property is not affected by the value of the Offset property.

This property returns both the date and the time component of a DateTimeOffset object, which makes it useful for performing DateTimeOffset to DateTime conversion. For example, if the DateTimeOffset object has a date and time value of "1/12/07 4:01pm + 7:30", the property returns a date of "1/12/07 4:01pm".

The value of the DateTime.Kind property of the returned DateTime object is DateTimeKind.Unspecified.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- LocalDateTime
- Converting Between DateTime and DateTimeOffset

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTimeOffset.Day Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the month represented by the current DateTimeOffset object.

C#

```
public int Day { get; }
```

Property Value

Int32

The day component of the current DateTimeOffset object, expressed as a value between 1 and 31.

Examples

The following example displays the day component of a DateTimeOffset object in three different ways:

- By retrieving the value of the Day property.
- By calling the ToString(String) method with the "d" format specifier.
- By calling the ToString(String) method with the "dd" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2007, 5, 1, 16, 35, 0,
                                            DateTimeOffset.Now.Offset);
Console.WriteLine("The day component of {0} is {1}.",
                  theTime, theTime.Day);

Console.WriteLine("The day component of {0} is{1}.",
                  theTime, theTime.ToString(" d"));

Console.WriteLine("The day component of {0} is {1}.",
                  theTime, theTime.ToString("dd"));
```

```
// The example produces the following output:
//    The day component of 5/1/2007 4:35:00 PM -08:00 is 1.
//    The day component of 5/1/2007 4:35:00 PM -08:00 is 1.
//    The day component of 5/1/2007 4:35:00 PM -08:00 is 01.
```

Remarks

The Day property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's day component by calling the ToString method with the "d" or "dd" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.DayOfWeek Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the week represented by the current DateTimeOffset object.

C#

```
public DayOfWeek DayOfWeek { get; }
```

Property Value

DayOfWeek

One of the enumeration values that indicates the day of the week of the current DateTimeOffset object.

Examples

The following example displays the weekday name of the first day of each month of the year 2008.

C#

```
DateTimeOffset startOfMonth = new DateTimeOffset(2008, 1, 1, 0, 0, 0,
                                                 DateTimeOffset.Now.Offset);
int year = startOfMonth.Year;
do
{
   Console.WriteLine("{0:MMM d, yyyy} is a {1}.", startOfMonth,
startOfMonth.DayOfWeek);
   startOfMonth = startOfMonth.AddMonths(1);
}
while (startOfMonth.Year == year);
// This example writes the following output to the console:
//    Jan 1, 2008 is a Tuesday.
//    Feb 1, 2008 is a Friday.
//    Mar 1, 2008 is a Saturday.
//    Apr 1, 2008 is a Tuesday.
//    May 1, 2008 is a Thursday.
```

```
//    Jun 1, 2008 is a Sunday.
//    Jul 1, 2008 is a Tuesday.
//    Aug 1, 2008 is a Friday.
//    Sep 1, 2008 is a Monday.
//    Oct 1, 2008 is a Wednesday.
//    Nov 1, 2008 is a Saturday.
//    Dec 1, 2008 is a Monday.
```

Remarks

The value of the constants in the DayOfWeek enumeration ranges from DayOfWeek.Sunday to DayOfWeek.Saturday. If cast to an integer, its value ranges from zero (which indicates DayOfWeek.Sunday) to six (which indicates DayOfWeek.Saturday).

You can also display the weekday name of a particular date by using the "D" format specifier or the "dddd" custom format specifier. For example:

C#

```
DateTimeOffset displayDate = new DateTimeOffset(2008, 1, 1, 13, 18, 00,
                                                DateTimeOffset.Now.Offset);
Console.WriteLine("{0:D}", displayDate);  // Output: Tuesday, January 01,
2008
Console.WriteLine("{0:d} is a {0:dddd}.",
                  displayDate);           // Output: 1/1/2008 is a Tuesday.
```

Note that the string returned by calling the `ToString` method of the DayOfWeek enumeration member that is returned by this property is not localized. To extract a string that contains the weekday name of the current culture or of a specific culture, call the ToString method with the "dddd" custom format specifier. For example, the following code displays the weekday name for a date using the `fr-fr` culture.

C#

```
DateTimeOffset thisDate = new DateTimeOffset(2007, 6, 1, 6, 15, 0,
                                             DateTimeOffset.Now.Offset);
string weekdayName = thisDate.ToString("dddd",
                                       new CultureInfo("fr-fr"));
Console.WriteLine(weekdayName);                  // Displays vendredi
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.DayOfYear Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the year represented by the current DateTimeOffset object.

C#

```
public int DayOfYear { get; }
```

Property Value

Int32

The day of the year of the current DateTimeOffset object, expressed as a value between 1 and 366.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTimeOffset.Hour Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hour component of the time represented by the current DateTimeOffset object.

C#

```
public int Hour { get; }
```

Property Value

Int32

The hour component of the current DateTimeOffset object. This property uses a 24-hour clock; the value ranges from 0 to 23.

Examples

The following example displays the hour component of a DateTimeOffset object in three different ways:

- By retrieving the value of the Hour property.
- By calling the ToString(String) method with the "H" format specifier.
- By calling the ToString(String) method with the "HH" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2008, 3, 1, 14, 15, 00,
                                            DateTimeOffset.Now.Offset);
Console.WriteLine("The hour component of {0} is {1}.",
                  theTime, theTime.Hour);

Console.WriteLine("The hour component of {0} is{1}.",
                  theTime, theTime.ToString(" H"));

Console.WriteLine("The hour component of {0} is {1}.",
                  theTime, theTime.ToString("HH"));
```

```
// The example produces the following output:
//    The hour component of 3/1/2008 2:15:00 PM -08:00 is 14.
//    The hour component of 3/1/2008 2:15:00 PM -08:00 is 14.
//    The hour component of 3/1/2008 2:15:00 PM -08:00 is 14.
```

Remarks

The Hour property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's hour component by calling the ToString method with the "H", or "HH" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.LocalDateTime Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime value that represents the local date and time of the current DateTimeOffset object.

C#

```
public DateTime LocalDateTime { get; }
```

Property Value

DateTime

The local date and time of the current DateTimeOffset object.

Examples

The following example illustrates several conversions of DateTimeOffset values to local times in the U.S. Pacific Standard Time zone. Note that the last three times are all ambiguous; the property maps all of them to a single date and time in the Pacific Standard Time zone.

C#

```
 DateTimeOffset dto;

 // Current time
 dto = DateTimeOffset.Now;
 Console.WriteLine(dto.LocalDateTime);
 // UTC time
 dto = DateTimeOffset.UtcNow;
 Console.WriteLine(dto.LocalDateTime);

// Transition to DST in local time zone occurs on 3/11/2007 at 2:00 AM
 dto = new DateTimeOffset(2007, 3, 11, 3, 30, 0, new TimeSpan(-7, 0, 0));
 Console.WriteLine(dto.LocalDateTime);
 dto = new DateTimeOffset(2007, 3, 11, 2, 30, 0, new TimeSpan(-7, 0, 0));
```

```
Console.WriteLine(dto.LocalDateTime);
// Invalid time in local time zone
dto = new DateTimeOffset(2007, 3, 11, 2, 30, 0, new TimeSpan(-8, 0, 0));
Console.WriteLine(TimeZoneInfo.Local.IsInvalidTime(dto.DateTime));
Console.WriteLine(dto.LocalDateTime);

// Transition from DST in local time zone occurs on 11/4/07 at 2:00 AM
// This is an ambiguous time
dto = new DateTimeOffset(2007, 11, 4, 1, 30, 0, new TimeSpan(-7, 0, 0));
Console.WriteLine(TimeZoneInfo.Local.IsAmbiguousTime(dto));
Console.WriteLine(dto.LocalDateTime);
dto = new DateTimeOffset(2007, 11, 4, 2, 30, 0, new TimeSpan(-7, 0, 0));
Console.WriteLine(TimeZoneInfo.Local.IsAmbiguousTime(dto));
Console.WriteLine(dto.LocalDateTime);
// This is also an ambiguous time
dto = new DateTimeOffset(2007, 11, 4, 1, 30, 0, new TimeSpan(-8, 0, 0));
Console.WriteLine(TimeZoneInfo.Local.IsAmbiguousTime(dto));
Console.WriteLine(dto.LocalDateTime);
// If run on 3/8/2007 at 4:56 PM, the code produces the following
// output:
//    3/8/2007 4:56:03 PM
//    3/8/2007 4:56:03 PM
//    3/11/2007 3:30:00 AM
//    3/11/2007 1:30:00 AM
//    True
//    3/11/2007 3:30:00 AM
//    True
//    11/4/2007 1:30:00 AM
//    11/4/2007 1:30:00 AM
//    True
//    11/4/2007 1:30:00 AM
```

Remarks

If necessary, the LocalDateTime property converts the current DateTimeOffset object's date and time to the local system's date and time. The conversion is a two-step operation:

1. The property converts the current DateTimeOffset object's time to Coordinated Universal Time (UTC).

2. The property then converts UTC to local time.

There are no invalid times and ambiguous times are mapped to the local zone's standard time. (However, the conversion can create an anomaly: if a DateTimeOffset value that is derived from the local computer reflects an ambiguous date and time, that value can be converted to UTC and then back to a local time that differs from the original time.) The property applies any adjustment rules in the local time zone when it performs this conversion.

This property returns both the date and the time component of a DateTimeOffset object, which makes it useful for DateTimeOffset to DateTime conversion. In addition to performing any necessary time conversion, this property differs from the DateTime property by setting the value of the Kind property of the returned DateTime object to DateTimeKind.Local.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- DateTime

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Microsecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the microsecond component of the time represented by the current DateTimeOffset object.

```
C#
public int Microsecond { get; }
```

Property Value

Int32

Remarks

If you rely on properties such as Now or UtcNow to accurately track the number of elapsed microseconds, the precision of the time's microseconds component depends on the resolution of the system clock. On Windows NT 3.5 and later, and Windows Vista operating systems, the clock's resolution is approximately 10000-15000 microseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

DateTimeOffset.Millisecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the millisecond component of the time represented by the current DateTimeOffset object.

C#

```
public int Millisecond { get; }
```

Property Value

Int32

The millisecond component of the current DateTimeOffset object, expressed as an integer between 0 and 999.

Examples

The following example displays the number of milliseconds of a DateTimeOffset object by using a custom format specifier and by directly accessing the Millisecond property.

C#

```
DateTimeOffset date1 = new DateTimeOffset(2008, 3, 5, 5, 45, 35, 649,
                                  new TimeSpan(-7, 0, 0));
Console.WriteLine("Milliseconds value of {0} is {1}.",
                  date1.ToString("MM/dd/yyyy hh:mm:ss.fff"),
                  date1.Millisecond);
// The example produces the following output:
//
// Milliseconds value of 03/05/2008 05:45:35.649 is 649.
```

Remarks

You can also create a string representation of a DateTimeOffset object's millisecond component by calling the ToString method with the "fff" custom format specifier.

If you rely on properties such as Now or UtcNow to accurately track the number of elapsed milliseconds, the precision of the time's millisecond component depends on the resolution of the system clock. On Windows NT 3.5 and later, and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Minute Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minute component of the time represented by the current DateTimeOffset object.

C#

```
public int Minute { get; }
```

Property Value

Int32

The minute component of the current DateTimeOffset object, expressed as an integer between 0 and 59.

Examples

The following example displays the minute component of a DateTimeOffset object in three different ways:

- By retrieving the value of the Minute property.
- By calling the ToString(String) method with the "m" format specifier.
- By calling the ToString(String) method with the "mm" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2008, 5, 1, 10, 3, 0,
                                            DateTimeOffset.Now.Offset);
Console.WriteLine("The minute component of {0} is {1}.",
                  theTime, theTime.Minute);

Console.WriteLine("The minute component of {0} is{1}.",
                  theTime, theTime.ToString(" m"));
```

```
Console.WriteLine("The minute component of {0} is {1}.",
                  theTime, theTime.ToString("mm"));
// The example produces the following output:
//    The minute component of 5/1/2008 10:03:00 AM -08:00 is 3.
//    The minute component of 5/1/2008 10:03:00 AM -08:00 is 3.
//    The minute component of 5/1/2008 10:03:00 AM -08:00 is 03.
```

Remarks

The Minute property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's minute component by calling the ToString method with the "m", or "mm" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Month Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the month component of the date represented by the current DateTimeOffset object.

C#

```
public int Month { get; }
```

Property Value

Int32

The month component of the current DateTimeOffset object, expressed as an integer between 1 and 12.

Examples

The following example displays the month component of a DateTimeOffset value in three different ways:

- By retrieving the value of the Month property.
- By calling the ToString(String) method with the "M" format specifier.
- By calling the ToString(String) method with the "MM" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2008, 9, 7, 11, 25, 0,
                                            DateTimeOffset.Now.Offset);
Console.WriteLine("The month component of {0} is {1}.",
                  theTime, theTime.Month);

Console.WriteLine("The month component of {0} is{1}.",
                  theTime, theTime.ToString(" M"));
```

```
Console.WriteLine("The month component of {0} is {1}.",
                  theTime, theTime.ToString("MM"));
// The example produces the following output:
//    The month component of 9/7/2008 11:25:00 AM -08:00 is 9.
//    The month component of 9/7/2008 11:25:00 AM -08:00 is 9.
//    The month component of 9/7/2008 11:25:00 AM -08:00 is 09.
```

Remarks

The Month property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's month component by calling the ToString method with the "M" or "MM" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Nanosecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the nanosecond component of the time represented by the current DateTimeOffset object.

C#

```
public int Nanosecond { get; }
```

Property Value

Int32

Remarks

If you rely on properties such as Now or UtcNow to accurately track the number of elapsed nanosecond, the precision of the time's nanosecond component depends on the resolution of the system clock. On Windows NT 3.5 and later, and Windows Vista operating systems, the clock's resolution is approximately 10000000-15000000 nanoseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTimeOffset.Now Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTimeOffset object that is set to the current date and time on the current computer, with the offset set to the local time's offset from Coordinated Universal Time (UTC).

C#

```
public static DateTimeOffset Now { get; }
```

Property Value

DateTimeOffset

A DateTimeOffset object whose date and time is the current local time and whose offset is the local time zone's offset from Coordinated Universal Time (UTC).

Examples

The following example uses the Now property to retrieve the current date and time and displays it by using each of the standard date and time format strings supported by the DateTimeOffset type.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] fmtStrings = { "d", "D", "f", "F", "g", "G", "M",
                              "R", "s", "t", "T", "u", "y" };

      DateTimeOffset value = DateTimeOffset.Now;
      // Display date in default format.
      Console.WriteLine(value);
```

```
      Console.WriteLine();

      // Display date using each of the specified formats.
      foreach (var fmtString in fmtStrings)
         Console.WriteLine("{0} --> {1}",
                           fmtString, value.ToString(fmtString));
   }
}
// The example displays output similar to the following:
//    11/19/2012 10:57:11 AM -08:00
//
//    d --> 11/19/2012
//    D --> Monday, November 19, 2012
//    f --> Monday, November 19, 2012 10:57 AM
//    F --> Monday, November 19, 2012 10:57:11 AM
//    g --> 11/19/2012 10:57 AM
//    G --> 11/19/2012 10:57:11 AM
//    M --> November 19
//    R --> Mon, 19 Nov 2012 18:57:11 GMT
//    s --> 2012-11-19T10:57:11
//    t --> 10:57 AM
//    T --> 10:57:11 AM
//    u --> 2012-11-19 18:57:11Z
//    y --> November, 2012
```

The following example uses the Now and Millisecond properties to determine the resolution of the system clock. It displays the time only when the value of its millisecond component has changed.

C#

```
DateTimeOffset dto;
int ctr = 0;
int ms = 0;
do {
   dto = DateTimeOffset.Now;
   if (dto.Millisecond != ms)
   {
      ms = dto.Millisecond;
      Console.WriteLine("{0}:{1:d3} ms. {2}",
                        dto.ToString("M/d/yyyy h:mm:ss"),
                        ms, dto.ToString("zzz"));
      ctr++;
   }
} while (ctr < 100);
```

Remarks

The precision of the current local time's millisecond component depends on the resolution of the system clock. On Windows NT 3.5 and later, and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UtcNow

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Offset Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time's offset from Coordinated Universal Time (UTC).

C#

```
public TimeSpan Offset { get; }
```

Property Value

TimeSpan

The difference between the current DateTimeOffset object's time value and Coordinated Universal Time (UTC).

Examples

The following example uses the Offset property to display the local time's difference from Coordinated Universal Time (UTC).

C#

```
DateTimeOffset localTime = DateTimeOffset.Now;
Console.WriteLine("The local time zone is {0} hours and {1} minutes {2} than
UTC.",
                  Math.Abs(localTime.Offset.Hours),
                  localTime.Offset.Minutes,
                  localTime.Offset.Hours < 0 ? "earlier" : "later");
// The example displays output similar to the following for a system in the
// U.S. Pacific Standard Time zone:
//       The local time zone is 8 hours and 0 minutes earlier than UTC.
```

Remarks

The value of the Hours property of the returned TimeSpan object can range from -14 hours to 14 hours.

The value of the Offset property is precise to the minute.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Second Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the second component of the clock time represented by the current DateTimeOffset object.

C#

```
public int Second { get; }
```

Property Value

Int32

The second component of the DateTimeOffset object, expressed as an integer value between 0 and 59.

Examples

The following example displays the second component of a DateTimeOffset object in three different ways:

- By retrieving the value of the Second property.
- By calling the ToString(String) method with the "s" format specifier.
- By calling the ToString(String) method with the "ss" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2008, 6, 12, 21, 16, 32,
                             DateTimeOffset.Now.Offset);
Console.WriteLine("The second component of {0} is {1}.",
                  theTime, theTime.Second);

Console.WriteLine("The second component of {0} is{1}.",
                  theTime, theTime.ToString(" s"));
```

```
Console.WriteLine("The second component of {0} is {1}.",
                  theTime, theTime.ToString("ss"));
// The example produces the following output:
//    The second component of 6/12/2008 9:16:32 PM -07:00 is 32.
//    The second component of 6/12/2008 9:16:32 PM -07:00 is 32.
//    The second component of 6/12/2008 9:16:32 PM -07:00 is 32.
```

Remarks

The Second property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's second component by calling the ToString method with the "s", or "ss" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Ticks Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of ticks that represents the date and time of the current DateTimeOffset object in clock time.

C#

```
public long Ticks { get; }
```

Property Value

Int64

The number of ticks in the DateTimeOffset object's clock time.

Examples

The following example initializes a DateTimeOffset object by approximating the number of ticks in the date July 1, 2008 1:23:07. It then displays the date and the number of ticks represented by that date to the console.

C#

```
// Attempt to initialize date to number of ticks
// in July 1, 2008 1:23:07.
//
// There are 10,000,000 100-nanosecond intervals in a second
const long NSPerSecond = 10000000;
long ticks;
ticks = 7 * NSPerSecond;                        // Ticks in a 7 seconds
ticks += 23 * 60 * NSPerSecond;                 // Ticks in 23 minutes
ticks += 1 * 60 * 60 * NSPerSecond;             // Ticks in 1 hour
ticks += 60 * 60 * 24 * NSPerSecond;            // Ticks in 1 day
ticks += 181 * 60 * 60 * 24 * NSPerSecond;      // Ticks in 6 months
ticks += 2007 * 60 * 60 * 24 * 365L * NSPerSecond;   // Ticks in 2007 years
ticks += 486 * 60 * 60 * 24 * NSPerSecond;      // Adjustment for leap years
DateTimeOffset dto = new DateTimeOffset(
                         ticks,
```

```
                                DateTimeOffset.Now.Offset);
Console.WriteLine("There are {0:n0} ticks in {1}.",
                  dto.Ticks,
                  dto.ToString());
// The example displays the following output:
//       There are 633,504,721,870,000,000 ticks in 7/1/2008 1:23:07 AM
-08:00.
```

Remarks

The Ticks property is not affected by the value of the Offset property.

The value of the Ticks property represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight on January 1, 0001 (the value of MinValue). It does not include ticks that would be added by leap seconds. A nanosecond is one billionth of a second; there are ten million ticks in a second. The value of the Ticks property ranges from `DateTimeOffset.MinValue.Ticks` to `DateTimeOffset.MaxValue.Ticks`.

You can assign the number of ticks to a DateTimeOffset object by using the DateTimeOffset(Int64, TimeSpan) constructor overload.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UtcDateTime

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTimeOffset.TimeOfDay Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time of day for the current DateTimeOffset object.

C#

```
public TimeSpan TimeOfDay { get; }
```

Property Value

TimeSpan

The time interval of the current date that has elapsed since midnight.

Examples

The following example uses the TimeOfDay property to extract the time and display it to the console.

C#

```
DateTimeOffset currentDate = new DateTimeOffset(2008, 5, 10, 5, 32, 16,
                                                DateTimeOffset.Now.Offset);
TimeSpan currentTime = currentDate.TimeOfDay;
Console.WriteLine("The current time is {0}.", currentTime.ToString());
// The example produces the following output:
//       The current time is 05:32:16.
```

Remarks

The TimeOfDay property is not affected by the value of the Offset property.

The TimeOfDay property returns the time component of a DateTimeOffset object in the form of a TimeSpan object. It is equivalent to the DateTime.TimeOfDay property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.TotalOffsetMinutes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time's offset from Coordinated Universal Time (UTC) in minutes.

C#

```
public int TotalOffsetMinutes { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.UtcDateTime Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTime value that represents the Coordinated Universal Time (UTC) date and time of the current DateTimeOffset object.

C#

```
public DateTime UtcDateTime { get; }
```

Property Value

DateTime

The Coordinated Universal Time (UTC) date and time of the current DateTimeOffset object.

Examples

The following example shows how to use of the UtcDateTime property to display a DateTimeOffset value and its corresponding UTC time.

C#

```
DateTimeOffset offsetTime = new DateTimeOffset(2007, 11, 25, 11, 14, 00,
                            new TimeSpan(3, 0, 0));
Console.WriteLine("{0} is equivalent to {1} {2}",
                  offsetTime.ToString(),
                  offsetTime.UtcDateTime.ToString(),
                  offsetTime.UtcDateTime.Kind.ToString());
// The example displays the following output:
//       11/25/2007 11:14:00 AM +03:00 is equivalent to 11/25/2007 8:14:00
AM Utc
```

Remarks

The UtcDateTime property performs a dual conversion:

- It converts the date and time of the current DateTimeOffset object to Coordinated Universal Time (UTC). The conversion is performed by subtracting the value of the Offset property from the date and time of the current DateTimeOffset object.
- It converts the DateTimeOffset value to a DateTime value.

The Kind property of the returned DateTime value is set to DateTimeKind.Utc.

Retrieving the value of the UtcDateTime property is equivalent to calling the current DateTimeOffset object's `ToUniversalTime.DateTime` property, except that the Kind property of the latter DateTime value is DateTimeKind.Unspecified.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.UtcNow Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a DateTimeOffset object whose date and time are set to the current Coordinated Universal Time (UTC) date and time and whose offset is Zero.

C#

```
public static DateTimeOffset UtcNow { get; }
```

Property Value

DateTimeOffset

An object whose date and time is the current Coordinated Universal Time (UTC) and whose offset is Zero.

Examples

The following example illustrates the relationship between Coordinated Universal Time (UTC) and local time.

C#

```
DateTimeOffset localTime = DateTimeOffset.Now;
DateTimeOffset utcTime = DateTimeOffset.UtcNow;

Console.WriteLine("Local Time:          {0}", localTime.ToString("T"));
Console.WriteLine("Difference from UTC: {0}", localTime.Offset.ToString());
Console.WriteLine("UTC:                 {0}", utcTime.ToString("T"));
// If run on a particular date at 1:19 PM, the example produces
// the following output:
//    Local Time:          1:19:43 PM
//    Difference from UTC: -07:00:00
//    UTC:                 8:19:43 PM
```

Remarks

The UtcNow property computes the current Universal Coordinated Time (UTC) based on the local system's clock time and an offset defined by the local system's time zone.

The precision of the current UTC time's millisecond component depends on the resolution of the system clock. On Windows NT 3.5 and later, and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Now

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.UtcTicks Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of ticks that represents the date and time of the current DateTimeOffset object in Coordinated Universal Time (UTC).

```
C#
public long UtcTicks { get; }
```

Property Value

Int64

The number of ticks in the DateTimeOffset object's Coordinated Universal Time (UTC).

Remarks

The value of the UtcTicks property represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight on January 1, 0001 (the value of MinValue). It does not include ticks that would be added by leap seconds. A nanosecond is one billionth of a second; there are ten million ticks in a second. The value of the UtcTicks property ranges from `DateTimeOffset.MinValue.Ticks` to `DateTimeOffset.MaxValue.Ticks`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.Year Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the year component of the date represented by the current DateTimeOffset object.

C#

```
public int Year { get; }
```

Property Value

Int32

The year component of the current DateTimeOffset object, expressed as an integer value between 0 and 9999.

Examples

The following example displays the year component of a DateTimeOffset value in four different ways:

- By retrieving the value of the Year property.
- By calling the ToString(String) method with the "y" format specifier.
- By calling the ToString(String) method with the "yy" format specifier.
- By calling the ToString(String) method with the "yyyy" format specifier.

C#

```
DateTimeOffset theTime = new DateTimeOffset(2008, 2, 17, 9, 0, 0,
                             DateTimeOffset.Now.Offset);
Console.WriteLine("The year component of {0} is {1}.",
                  theTime, theTime.Year);

Console.WriteLine("The year component of {0} is{1}.",
                  theTime, theTime.ToString(" y"));
```

```
Console.WriteLine("The year component of {0} is {1}.",
                  theTime, theTime.ToString("yy"));

Console.WriteLine("The year component of {0} is {1}.",
                  theTime, theTime.ToString("yyyy"));
// The example produces the following output:
//    The year component of 2/17/2008 9:00:00 AM -07:00 is 2008.
//    The year component of 2/17/2008 9:00:00 AM -07:00 is 8.
//    The year component of 2/17/2008 9:00:00 AM -07:00 is 08.
//    The year component of 2/17/2008 9:00:00 AM -07:00 is 2008.
```

Remarks

The Year property is not affected by the value of the Offset property.

You can also create a string representation of a DateTimeOffset object's year component by calling the ToString method with the "y", "yy", or "yyyy" custom format specifiers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Add(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified time interval to the value of this instance.

```
C#

public DateTimeOffset Add (TimeSpan timeSpan);
```

Parameters

`timeSpan` TimeSpan

A TimeSpan object that represents a positive or a negative time interval.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the time interval represented by `timeSpan`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

The following example creates an array of TimeSpan objects that represent the flight times between destinations. The Add method then adds these times to a

DateTimeOffset object that represents a flight's initial takeoff time. The result reflects the scheduled arrival time at each destination.

C#

```
DateTimeOffset takeOff = new DateTimeOffset(2007, 6, 1, 7, 55, 0,
                             new TimeSpan(-5, 0, 0));
DateTimeOffset currentTime = takeOff;
TimeSpan[] flightTimes = new TimeSpan[]
                  {new TimeSpan(2, 25, 0), new TimeSpan(1, 48, 0)};
Console.WriteLine("Takeoff is scheduled for {0:d} at {0:T}.",
                  takeOff);
for (int ctr = flightTimes.GetLowerBound(0);
     ctr <= flightTimes.GetUpperBound(0); ctr++)
{
   currentTime = currentTime.Add(flightTimes[ctr]);
   Console.WriteLine("Destination #{0} at {1}.", ctr + 1, currentTime);
}
```

Remarks

You can use the Add method to add more than one kind of time interval (days, hours, minutes, seconds, or milliseconds) in a single operation. This method's behavior is identical to the addition operator. The DateTimeOffset structure also supports specialized addition methods (such as AddDays, AddHours, and AddMinutes) for each time interval.

> ⓘ **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `timeSpan` to its date and time.

The Add method does not affect the value of the current DateTimeOffset object's Offset property.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the Add method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

If the `timeSpan` parameter is `null`, this method returns the value of the original DateTimeOffset object unchanged.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Addition(DateTimeOffset, TimeSpan)
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.AddDays(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of whole and fractional days to the value of this instance.

```
C#

public DateTimeOffset AddDays (double days);
```

Parameters

days Double

A number of whole and fractional days. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of days represented by `days`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

The following example uses the AddDays method to list the dates that fall on Monday, the start of the work week, in March 2008.

C#

```
DateTimeOffset workDay = new DateTimeOffset(2008, 3, 1, 9, 0, 0,
                    DateTimeOffset.Now.Offset);
int month = workDay.Month;
// Start with the first Monday of the month
if (workDay.DayOfWeek != DayOfWeek.Monday)
{
   if (workDay.DayOfWeek == DayOfWeek.Sunday)
      workDay = workDay.AddDays(1);
   else
      workDay = workDay.AddDays(8 - (int)workDay.DayOfWeek);
}
Console.WriteLine("Beginning of Work Week In {0:MMMM} {0:yyyy}:", workDay);
// Add one week to the current date
do
{
   Console.WriteLine("   {0:dddd}, {0:MMMM}{0: d}", workDay);
   workDay = workDay.AddDays(7);
} while (workDay.Month == month);
// The example produces the following output:
//    Beginning of Work Week In March 2008:
//       Monday, March 3
//       Monday, March 10
//       Monday, March 17
//       Monday, March 24
//       Monday, March 31
```

Remarks

The fractional part of the `days` parameter is the fractional part of a day. For example, 4.5 is equivalent to 4 days, 12 hours, 0 minutes, 0 seconds, 0 milliseconds.

In .NET 6 and earlier versions, the `days` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `days` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `days` to its date and time.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddDays method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Converting time intervals of less than a day to a fraction can involve a loss of precision. If this is problematic, you can use the Add method, which enables you to specify more than one kind of time interval in a single method call and eliminates the need to convert time intervals to fractional parts of a day.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.AddHours(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of whole and fractional hours to the value of this instance.

```
C#

public DateTimeOffset AddHours (double hours);
```

Parameters

hours Double

A number of whole and fractional hours. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of hours represented by `hours`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

The following example uses the AddHours method to list the start times of work shifts for a particular week at an office that has two eight-hour shifts per day.

C#

```
const int SHIFT_LENGTH = 8;

DateTimeOffset startTime = new DateTimeOffset(2007, 8, 6, 0, 0, 0,
                      DateTimeOffset.Now.Offset);
DateTimeOffset startOfShift = startTime.AddHours(SHIFT_LENGTH);

Console.WriteLine("Shifts for the week of {0:D}", startOfShift);
do
{
   // Exclude third shift
   if (startOfShift.Hour > 6)
      Console.WriteLine("   {0:d} at {0:T}", startOfShift);

   startOfShift = startOfShift.AddHours(SHIFT_LENGTH);
} while (startOfShift.DayOfWeek != DayOfWeek.Saturday &
            startOfShift.DayOfWeek != DayOfWeek.Sunday);
// The example produces the following output:
//
//    Shifts for the week of Monday, August 06, 2007
//       8/6/2007 at 8:00:00 AM
//       8/6/2007 at 4:00:00 PM
//       8/7/2007 at 8:00:00 AM
//       8/7/2007 at 4:00:00 PM
//       8/8/2007 at 8:00:00 AM
//       8/8/2007 at 4:00:00 PM
//       8/9/2007 at 8:00:00 AM
//       8/9/2007 at 4:00:00 PM
//       8/10/2007 at 8:00:00 AM
//       8/10/2007 at 4:00:00 PM
```

Remarks

The fractional part of the `hours` parameter is the fractional part of an hour. For example, 4.5 is equivalent to 4 hours, 30 minutes, 0 seconds, 0 milliseconds.

In .NET 6 and earlier versions, the `hours` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `hours` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

Note

This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `hours` to its date and time.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddHours method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Converting time intervals of less than an hour to a fraction can involve a loss of precision. (For example, one minute is 0.01666 of an hour.) If this is problematic, you can use the Add method, which enables you to specify more than one kind of time interval in a single method call and eliminates the need to convert time intervals to fractional parts of an hour.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.AddMicroseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of microseconds to the value of this instance.

```
C#

public DateTimeOffset AddMicroseconds (double microseconds);
```

Parameters

`microseconds` Double

A number of whole and fractional microseconds. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of whole microseconds represented by `microseconds`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

The fractional part of value is the fractional part of a microsecond. For example, 4.5 is equivalent to 4 microseconds and 50 ticks, where one microseconds = 10 ticks. However, `microseconds` is rounded to the nearest microsecond; all values of .5 or greater are rounded up.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddMicroseconds(Double) method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Add Milliseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of milliseconds to the value of this instance.

```
C#

public DateTimeOffset AddMilliseconds (double milliseconds);
```

Parameters

`milliseconds` Double

A number of whole and fractional milliseconds. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of whole milliseconds represented by `milliseconds`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

The fractional part of the `milliseconds` parameter is the fractional part of a millisecond. For example, 4.5 is equivalent to 4 milliseconds and 5000 ticks, where one millisecond equals 10,000 ticks. However, in .NET 6 and earlier versions, `milliseconds` is rounded to the nearest millisecond, and all values of .5 or greater are rounded up. In .NET 7 and later versions, the full Double precision of the `milliseconds` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `milliseconds` to its date and time.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddMilliseconds method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTimeOffset.AddMinutes(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of whole and fractional minutes to the value of this instance.

```
C#

public DateTimeOffset AddMinutes (double minutes);
```

Parameters

`minutes` Double

A number of whole and fractional minutes. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of minutes represented by `minutes`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

The fractional part of the `minutes` parameter is the fractional part of a minute. For example, 4.5 is equivalent to 4 minutes, 30 seconds, 0 milliseconds.

In .NET 6 and earlier versions, the `minutes` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `minutes` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

> ⓘ **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `minutes` to its date and time.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddMinutes method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Converting time intervals of less than a minute to a fraction can involve a loss of precision. (For example, one second is 0.01666 of a minute.) If this is problematic, you can use the Add method, which enables you to specify more than one kind of time interval in a single method call and eliminates the need to convert time intervals to fractional parts of a minute.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.AddMonths(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of months to the value of this instance.

```
C#

public DateTimeOffset AddMonths (int months);
```

Parameters

`months` Int32

A number of whole months. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of months represented by `months`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

The following example uses the AddMonths method to display the start date of each quarter of the year 2007.

C#

```
DateTimeOffset quarterDate = new DateTimeOffset(2007, 1, 1, 0, 0, 0,
                                 DateTimeOffset.Now.Offset);
for (int ctr = 1; ctr <= 4; ctr++)
{
   Console.WriteLine("Quarter {0}: {1:MMMM d}", ctr, quarterDate);
   quarterDate = quarterDate.AddMonths(3);
}
// This example produces the following output:
//       Quarter 1: January 1
//       Quarter 2: April 1
//       Quarter 3: July 1
//       Quarter 4: October 1
```

Remarks

Unlike most of the other methods that add a single time interval unit (such as minutes or days) to a date and time value, AddMonths does not enable you to add fractional parts of a month. To add a time that consists of other time units in addition to months to a DateTimeOffset object, use the Add method.

> ⓘ **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `months` to its date and time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.AddSeconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of whole and fractional seconds to the value of this instance.

```
C#

public DateTimeOffset AddSeconds (double seconds);
```

Parameters

`seconds` Double

A number of whole and fractional seconds. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of seconds represented by `seconds`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

The fractional part of the `seconds` parameter is the fractional part of a second. The value of fractional parts of a second are shown in the following table.

Expand table

Second value	Equivalent
.1 second	100 milliseconds
.01 second	10 milliseconds
.001 second	1 millisecond

In .NET 6 and earlier versions, the `seconds` parameter is rounded to the nearest millisecond. In .NET 7 and later versions, the full Double precision of the `seconds` parameter is used. However, due to the inherent imprecision of floating point math, the resulting precision will vary.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `seconds` to its date and time.

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddSeconds method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.AddTicks(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of ticks to the value of this instance.

```
C#

public DateTimeOffset AddTicks (long ticks);
```

Parameters

`ticks` Int64

A number of 100-nanosecond ticks. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of ticks represented by `ticks`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

A tick represents 100 nanoseconds, or one-ten millionth of a second. Because a tick is a very small and precise time interval, any larger time unit can be expressed as a number of ticks without a loss of precision.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `ticks` to its date and time.

The following table lists the number of ticks in a particular time interval.

Expand table

Time interval	Number of ticks
Second	10,000,000
Minute	600,000,000
Hour	36,000,000,000
Day	864,000,000,000
Week	6,048,000,000,000
Month	Depends on number of days in the month.
Non-leap year	315,360,000,000,000
Leap year	316,224,000,000,000

Because a DateTimeOffset object does not represent the date and time in a specific time zone, the AddTicks method does not consider a particular time zone's adjustment rules when it performs date and time arithmetic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.AddYears(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new DateTimeOffset object that adds a specified number of years to the value of this instance.

```
C#

public DateTimeOffset AddYears (int years);
```

Parameters

years Int32

A number of years. The number can be negative or positive.

Returns

DateTimeOffset

An object whose value is the sum of the date and time represented by the current DateTimeOffset object and the number of years represented by `years`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

In the United States, driver's licenses cannot be issued to persons under 16 years of age. The following example displays the latest possible date on which a person must be born in order to legally be issued a driver's license.

C#

```
const int minimumAge = 16;
DateTimeOffset dateToday = DateTimeOffset.Now;
DateTimeOffset latestBirthday = dateToday.AddYears(-1 * minimumAge);
Console.WriteLine("To possess a driver's license, you must have been born on
or before {0:d}.",
                  latestBirthday);
```

Remarks

Unlike most of the other methods that add a single time interval unit (such as minutes or days) to a date and time value, AddYears does not enable you to add fractional parts of a year. To add a time that consists of other time units in addition to years to a DateTimeOffset object, use the Add method.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by adding `years` to its date and time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two DateTimeOffset objects and indicates whether the first is earlier than the second, equal to the second, or later than the second.

C#

```
public static int Compare (DateTimeOffset first, DateTimeOffset second);
```

Parameters

`first` DateTimeOffset

The first object to compare.

`second` DateTimeOffset

The second object to compare.

Returns

Int32

A signed integer that indicates whether the value of the `first` parameter is earlier than, later than, or the same time as the value of the `second` parameter, as the following table shows.

Expand table

Return value	Meaning
Less than zero	`first` is earlier than `second`.
Zero	`first` is equal to `second`.

Return value	Meaning
Greater than zero	`first` is later than `second`.

Examples

The following example illustrates calls to the Compare method to compare DateTimeOffset objects.

C#

```
using System;

public class CompareTimes
{
   private enum TimeComparison
   {
      Earlier = -1,
      Same = 0,
      Later = 1
   };

   public static void Main()
   {
      DateTimeOffset firstTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                                 new TimeSpan(-7, 0, 0));

      DateTimeOffset secondTime = firstTime;
      Console.WriteLine("Comparing {0} and {1}: {2}",
                        firstTime, secondTime,
                        (TimeComparison) DateTimeOffset.Compare(firstTime,
secondTime));

      secondTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                       new TimeSpan(-6, 0, 0));
      Console.WriteLine("Comparing {0} and {1}: {2}",
                       firstTime, secondTime,
                       (TimeComparison) DateTimeOffset.Compare(firstTime,
secondTime));

      secondTime = new DateTimeOffset(2007, 9, 1, 8, 45, 0,
                       new TimeSpan(-5, 0, 0));
      Console.WriteLine("Comparing {0} and {1}: {2}",
                       firstTime, secondTime,
                       (TimeComparison) DateTimeOffset.Compare(firstTime,
secondTime));
      // The example displays the following output to the console:
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 6:45:00 AM
-07:00: Same
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 6:45:00 AM
-06:00: Later
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 8:45:00 AM
```

```
-05:00: Same
    }
}
```

Remarks

In performing the comparison, the method converts both the `first` and the `second` parameters to Coordinated Universal Time (UTC) before it performs the comparison. The method is equivalent to the following:

```C#
return DateTime.Compare(first.UtcDateTime, second.UtcDateTime);
```

In other words, the Compare method determines whether two DateTimeOffset objects represent a single point in time. It directly compares neither dates and times nor offsets.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.CompareTo(DateTimeOffset) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current DateTimeOffset object to a specified DateTimeOffset object and indicates whether the current object is earlier than, the same as, or later than the second DateTimeOffset object.

C#

```
public int CompareTo (DateTimeOffset other);
```

Parameters

other DateTimeOffset

An object to compare with the current DateTimeOffset object.

Returns

Int32

A signed integer that indicates the relationship between the current DateTimeOffset object and `other`, as the following table shows.

Expand table

Return Value	Description
Less than zero	The current DateTimeOffset object is earlier than `other`.
Zero	The current DateTimeOffset object is the same as `other`.
Greater than zero.	The current DateTimeOffset object is later than `other`.

Implements

CompareTo(T)

Examples

The following example illustrates calls to the CompareTo method to compare DateTimeOffset objects.

C#

```
using System;

public class CompareTimes
{
   private enum TimeComparison
   {
      Earlier = -1,
      Same = 0,
      Later = 1
   };

   public static void Main()
   {
      DateTimeOffset firstTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                                 new TimeSpan(-7, 0, 0));

      DateTimeOffset secondTime = firstTime;
      Console.WriteLine("Comparing {0} and {1}: {2}",
                        firstTime, secondTime,
                        (TimeComparison) firstTime.CompareTo(secondTime));

      secondTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                       new TimeSpan(-6, 0, 0));
      Console.WriteLine("Comparing {0} and {1}: {2}",
                       firstTime, secondTime,
                       (TimeComparison) firstTime.CompareTo(secondTime));

      secondTime = new DateTimeOffset(2007, 9, 1, 8, 45, 0,
                       new TimeSpan(-5, 0, 0));
      Console.WriteLine("Comparing {0} and {1}: {2}",
                       firstTime, secondTime,
                       (TimeComparison) firstTime.CompareTo(secondTime));
      // The example displays the following output to the console:
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 6:45:00 AM
-07:00: Same
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 6:45:00 AM
-06:00: Later
      //       Comparing 9/1/2007 6:45:00 AM -07:00 and 9/1/2007 8:45:00 AM
-05:00: Same
   }
}
```

Remarks

This method compares DateTimeOffset objects by comparing their UtcDateTime values; that is, it determines whether the two objects represent a single point in time, and indicates whether the current object is earlier than, later than, or the same as the `other` parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Deconstruct(DateOnly, TimeOnly, TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Deconstructs this DateTimeOffset instance by DateOnly, TimeOnly, and TimeSpan.

C#

```
public void Deconstruct (out DateOnly date, out TimeOnly time, out TimeSpan
offset);
```

Parameters

`date` DateOnly

When this method returns, represents the DateOnly value of this DateTimeOffset instance.

`time` TimeOnly

When this method returns, represents the TimeOnly value of this DateTimeOffset instance.

`offset` TimeSpan

When this method returns, represents the Offset value of this DateTimeOffset instance.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two DateTimeOffset objects are equal, or a DateTimeOffset object is equal to a specified object.

Overloads

Expand table

Equals(DateTimeOffset)	Determines whether the current DateTimeOffset object represents the same point in time as a specified DateTimeOffset object.
Equals(Object)	Determines whether a DateTimeOffset object represents the same point in time as a specified object.
Equals(DateTimeOffset, DateTimeOffset)	Determines whether two specified DateTimeOffset objects represent the same point in time.

Equals(DateTimeOffset)

Determines whether the current DateTimeOffset object represents the same point in time as a specified DateTimeOffset object.

C#

```
public bool Equals (DateTimeOffset other);
```

Parameters

`other` DateTimeOffset

An object to compare to the current DateTimeOffset object.

Returns

Boolean

`true` if both DateTimeOffset objects have the same UtcDateTime value; otherwise, `false`.

Implements

Equals(T)

Examples

The following example illustrates calls to the Equals(DateTimeOffset) method to test DateTimeOffset objects for equality with the current DateTimeOffset object.

C#

```
private static void CompareForEquality1()
{
   DateTimeOffset firstTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                              new TimeSpan(-7, 0, 0));

   DateTimeOffset secondTime = firstTime;
   Console.WriteLine("{0} = {1}: {2}",
                     firstTime, secondTime,
                     firstTime.Equals(secondTime));

   secondTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                    new TimeSpan(-6, 0, 0));
   Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
                    firstTime.Equals(secondTime));

   secondTime = new DateTimeOffset(2007, 9, 1, 8, 45, 0,
                    new TimeSpan(-5, 0, 0));
   Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
                    firstTime.Equals(secondTime));
   // The example displays the following output to the console:
   //      9/1/2007 6:45:00 AM -07:00 = 9/1/2007 6:45:00 AM -07:00: True
   //      9/1/2007 6:45:00 AM -07:00 = 9/1/2007 6:45:00 AM -06:00: False
   //      9/1/2007 6:45:00 AM -07:00 = 9/1/2007 8:45:00 AM -05:00: True
```

Remarks

Before it performs the comparison, this method converts the values of both DateTimeOffset objects to Coordinated Universal Time (UTC). The method is

equivalent to the following:

```
C#

return this.UtcDateTime == other.UtcDateTime;
```

In other words, the Equals(DateTimeOffset) method determines whether two DateTimeOffset objects represent a single point in time. It directly compares neither dates and times nor offsets. To determine whether two DateTimeOffset objects represent the same time and have the same offset value, use the EqualsExact method.

A DateTimeOffset object that is not `null` is considered to be later (or greater) than one that is `null`.

This overload of the Equals(DateTimeOffset) method implements the IEquatable<T>.Equals method. It offers slightly better performance than the DateTimeOffset.Equals(Object) overload because the `other` parameter does not have to be converted from an object.

See also

- EqualsExact(DateTimeOffset)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Determines whether a DateTimeOffset object represents the same point in time as a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare to the current DateTimeOffset object.

Returns

Boolean

`true` if the `obj` parameter is a DateTimeOffset object and represents the same point in time as the current DateTimeOffset object; otherwise, `false`.

Examples

The following example indicates whether the current DateTimeOffset object is equal to several other DateTimeOffset objects, as well as to a null reference and a DateTime object.

```
C#

private static void CompareForEquality2()
{
   DateTimeOffset firstTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                              new TimeSpan(-7, 0, 0));

   object secondTime = firstTime;
   Console.WriteLine("{0} = {1}: {2}",
                     firstTime, secondTime,
                     firstTime.Equals(secondTime));

   secondTime = new DateTimeOffset(2007, 9, 1, 6, 45, 0,
                    new TimeSpan(-6, 0, 0));
   Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
                    firstTime.Equals(secondTime));

   secondTime = new DateTimeOffset(2007, 9, 1, 8, 45, 0,
                    new TimeSpan(-5, 0, 0));
   Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
```

```
                    firstTime.Equals(secondTime));

    secondTime = null;
    Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
                    firstTime.Equals(secondTime));

    secondTime = new DateTime(2007, 9, 1, 6, 45, 00);
    Console.WriteLine("{0} = {1}: {2}",
                    firstTime, secondTime,
                    firstTime.Equals(secondTime));
    // The example displays the following output to the console:
    //       9/1/2007 6:45:00 AM -07:00 = 9/1/2007 6:45:00 AM -07:00: True
    //       9/1/2007 6:45:00 AM -07:00 = 9/1/2007 6:45:00 AM -06:00:
  False
    //       9/1/2007 6:45:00 AM -07:00 = 9/1/2007 8:45:00 AM -05:00: True
    //       9/1/2007 6:45:00 AM -07:00 = : False
    //       9/1/2007 6:45:00 AM -07:00 = 9/1/2007 6:45:00 AM: False
```

Remarks

Before it performs the comparison, this method converts the values of both the current DateTimeOffset object and the `obj` parameter to Coordinated Universal Time (UTC). The method is equivalent to the following:

```
C#

return this.UtcDateTime == ((DateTimeOffset) obj).UtcDateTime;
```

In other words, the DateTimeOffset.Equals(Object) method determines whether the current DateTimeOffset object and a specified object represent a single point in time. It directly compares neither dates and times nor offsets. To determine whether two DateTimeOffset objects represent the same time and have the same offset value, use the EqualsExact method.

If `obj` is `null`, or if the run-time type of `obj` is not DateTimeOffset, the method returns `false`.

See also

- EqualsExact(DateTimeOffset)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(DateTimeOffset, DateTimeOffset)

Determines whether two specified DateTimeOffset objects represent the same point in time.

```
C#

public static bool Equals (DateTimeOffset first, DateTimeOffset second);
```

Parameters

`first` DateTimeOffset

The first object to compare.

`second` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if the two DateTimeOffset objects have the same UtcDateTime value; otherwise, `false`.

Examples

The following example illustrates calls to the Equals(DateTimeOffset, DateTimeOffset) method to test various pairs of DateTimeOffset objects for equality.

```
C#
```

```
DateTimeOffset firstTime = new DateTimeOffset(2007, 11, 15, 11, 35, 00,
                                    DateTimeOffset.Now.Offset);
DateTimeOffset secondTime = firstTime;
Console.WriteLine("{0} = {1}: {2}",
                  firstTime, secondTime,
                  DateTimeOffset.Equals(firstTime, secondTime));

// The value of firstTime remains unchanged
secondTime = new DateTimeOffset(firstTime.DateTime,
             TimeSpan.FromHours(firstTime.Offset.Hours + 1));
Console.WriteLine("{0} = {1}: {2}",
                  firstTime, secondTime,
                  DateTimeOffset.Equals(firstTime, secondTime));

// value of firstTime remains unchanged
secondTime = new DateTimeOffset(firstTime.DateTime +
TimeSpan.FromHours(1),
                                TimeSpan.FromHours(firstTime.Offset.Hours
+ 1));
Console.WriteLine("{0} = {1}: {2}",
                  firstTime, secondTime,
                  DateTimeOffset.Equals(firstTime, secondTime));
 // The example produces the following output:
 //       11/15/2007 11:35:00 AM -07:00 = 11/15/2007 11:35:00 AM -07:00:
True
 //       11/15/2007 11:35:00 AM -07:00 = 11/15/2007 11:35:00 AM -06:00:
False
 //       11/15/2007 11:35:00 AM -07:00 = 11/15/2007 12:35:00 PM -06:00:
True
```

Remarks

Before it performs the comparison, this method converts both DateTimeOffset objects to Coordinated Universal Time (UTC). The method is equivalent to the following:

C#

```
return first.UtcDateTime == second.UtcDateTime;
```

In other words, the Equals(DateTimeOffset, DateTimeOffset) method determines whether the two DateTimeOffset objects represent a single point in time. It directly compares neither dates and times nor offsets. To determine whether two DateTimeOffset objects represent the same time and have the same offset value, use the EqualsExact method.

See also

- EqualsExact(DateTimeOffset)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Equals Exact(DateTimeOffset) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the current DateTimeOffset object represents the same time and has the same offset as a specified DateTimeOffset object.

C#

```
public bool EqualsExact (DateTimeOffset other);
```

Parameters

other DateTimeOffset

The object to compare to the current DateTimeOffset object.

Returns

Boolean

`true` if the current DateTimeOffset object and `other` have the same date and time value and the same Offset value; otherwise, `false`.

Examples

The following example illustrates the use of the EqualsExact method to compare similar DateTimeOffset objects.

C#

```
DateTimeOffset instanceTime = new DateTimeOffset(2007, 10, 31, 0, 0, 0,
                                DateTimeOffset.Now.Offset);

DateTimeOffset otherTime = instanceTime;
Console.WriteLine("{0} = {1}: {2}",
                  instanceTime, otherTime,
```

```
                    instanceTime.EqualsExact(otherTime));

otherTime = new DateTimeOffset(instanceTime.DateTime,
            TimeSpan.FromHours(instanceTime.Offset.Hours + 1));
Console.WriteLine("{0} = {1}: {2}",
                    instanceTime, otherTime,
                    instanceTime.EqualsExact(otherTime));

otherTime = new DateTimeOffset(instanceTime.DateTime +
TimeSpan.FromHours(1),
                TimeSpan.FromHours(instanceTime.Offset.Hours + 1));
Console.WriteLine("{0} = {1}: {2}",
                    instanceTime, otherTime,
                    instanceTime.EqualsExact(otherTime));
// The example produces the following output:
//       10/31/2007 12:00:00 AM -07:00 = 10/31/2007 12:00:00 AM -07:00: True
//       10/31/2007 12:00:00 AM -07:00 = 10/31/2007 12:00:00 AM -06:00:
False
//       10/31/2007 12:00:00 AM -07:00 = 10/31/2007 1:00:00 AM -06:00: False
```

Remarks

Because multiple time zones share a single offset, a return value of `true` does not guarantee that the current and the `other` object represent times in the same time zone.

Unlike the EqualsExact method, the overloads of the Equals method determine only whether two DateTimeOffset values represent a single point in time. They do not indicate that two values have the same date and time as well as the same offset.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.FromFileTime(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified Windows file time to an equivalent local time.

```
C#
public static DateTimeOffset FromFileTime (long fileTime);
```

Parameters

`fileTime` Int64

A Windows file time, expressed in ticks.

Returns

DateTimeOffset

An object that represents the date and time of `fileTime` with the offset set to the local time offset.

Exceptions

ArgumentOutOfRangeException

`filetime` is less than zero.

-or-

`filetime` is greater than `DateTimeOffset.MaxValue.Ticks`.

Examples

The following example uses the Windows API to retrieve the Windows file times for the WordPad executable.

C#

```
using System;
using System.IO;
using System.Runtime.InteropServices;

public struct FileTime
{
   public uint dwLowDateTime;
   public uint dwHighDateTime;

   public static implicit operator long(FileTime fileTime)
   {
      long returnedLong;
      // Convert 4 high-order bytes to a byte array
      byte[] highBytes = BitConverter.GetBytes(fileTime.dwHighDateTime);
      // Resize the array to 8 bytes (for a Long)
      Array.Resize(ref highBytes, 8);

      // Assign high-order bytes to first 4 bytes of Long
      returnedLong = BitConverter.ToInt64(highBytes, 0);
      // Shift high-order bytes into position
      returnedLong = returnedLong << 32;
      // Or with low-order bytes
      returnedLong = returnedLong | fileTime.dwLowDateTime;
      // Return long
      return returnedLong;
   }
}

public class FileTimes
{
   private const int OPEN_EXISTING = 3;
   private const int INVALID_HANDLE_VALUE = -1;

   [DllImport("Kernel32.dll", CharSet = CharSet.Unicode)]
   private static extern int CreateFile(string lpFileName,
                                        int dwDesiredAccess,
                                        int dwShareMode,
                                        int lpSecurityAttributes,
                                        int dwCreationDisposition,
                                        int dwFlagsAndAttributes,
                                        int hTemplateFile);

   [DllImport("Kernel32.dll")]
   private static extern bool GetFileTime(int hFile,
                                          out FileTime lpCreationTime,
                                          out FileTime lpLastAccessTime,
                                          out FileTime lpLastWriteTime);

   [DllImport("Kernel32.dll")]
   private static extern bool CloseHandle(int hFile);

   public static void Main()
```

```
   {
      // Open file %windir%\write.exe
      string winDir = Environment.SystemDirectory;
      if (! (winDir.EndsWith(Path.DirectorySeparatorChar.ToString())))
         winDir += Path.DirectorySeparatorChar;
      winDir += "write.exe";

      // Get file time using Windows API
      //
      // Open file
      int hFile = CreateFile(winDir, 0, 0, 0, OPEN_EXISTING, 0, 0);
      if (hFile == INVALID_HANDLE_VALUE)
      {
         Console.WriteLine("Unable to access {0}.", winDir);
      }
      else
      {
         FileTime creationTime, accessTime, writeTime;
         if (GetFileTime(hFile, out creationTime, out accessTime, out
writeTime))
         {
            CloseHandle(hFile);
            long fileCreationTime = (long) creationTime;
            long fileAccessTime = accessTime;
            long fileWriteTime = (long) writeTime;

            Console.WriteLine("File {0} Retrieved Using the Windows API:",
winDir);
            Console.WriteLine("   Created:     {0:d}",
DateTimeOffset.FromFileTime(fileCreationTime).ToString());
            Console.WriteLine("   Last Access: {0:d}",
DateTimeOffset.FromFileTime(fileAccessTime).ToString());
            Console.WriteLine("   Last Write:  {0:d}",
DateTimeOffset.FromFileTime(fileWriteTime).ToString());
            Console.WriteLine();
         }
      }

      // Get date and time, convert to file time, then convert back
      FileInfo fileInfo = new FileInfo(winDir);
      DateTimeOffset infoCreationTime, infoAccessTime, infoWriteTime;
      long ftCreationTime, ftAccessTime, ftWriteTime;

      // Get dates and times of file creation, last access, and last write
      infoCreationTime = fileInfo.CreationTime;
      infoAccessTime = fileInfo.LastAccessTime;
      infoWriteTime = fileInfo.LastWriteTime;
      // Convert values to file times
      ftCreationTime = infoCreationTime.ToFileTime();
      ftAccessTime = infoAccessTime.ToFileTime();
      ftWriteTime = infoWriteTime.ToFileTime();

      // Convert file times back to DateTimeOffset values
      Console.WriteLine("File {0} Retrieved Using a FileInfo Object:",
winDir);
```

```
        Console.WriteLine("   Created:     {0:d}",
DateTimeOffset.FromFileTime(ftCreationTime).ToString());
        Console.WriteLine("   Last Access: {0:d}",
DateTimeOffset.FromFileTime(ftAccessTime).ToString());
        Console.WriteLine("   Last Write:  {0:d}",
DateTimeOffset.FromFileTime(ftWriteTime).ToString());
    }
}
// The example produces the following output:
//    File C:\WINDOWS\system32\write.exe Retrieved Using the Windows API:
//       Created:     10/13/2005 5:26:59 PM -07:00
//       Last Access: 3/20/2007 2:07:00 AM -07:00
//       Last Write:  8/4/2004 5:00:00 AM -07:00
//
//    File C:\WINDOWS\system32\write.exe Retrieved Using a FileInfo Object:
//       Created:     10/13/2005 5:26:59 PM -07:00
//       Last Access: 3/20/2007 2:07:00 AM -07:00
//       Last Write:  8/4/2004 5:00:00 AM -07:00
```

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

A Windows file time is directly accessible through the Windows API by calling the `GetFileTime` function, which returns a `FILETIME` structure. The single function parameter is the handle of the file whose file time information is to be retrieved. The file handle is retrieved by calling the `CreateFile` function. The `FILETIME` structure's `dwHighDateTime` member contains the four high-order bytes of the file time, and its `dwLowDateTime` member contains the four low-order bytes. The example that follows illustrates how to retrieve Windows file time values and convert them to DateTimeOffset values.

Windows file time values can also be created from DateTime values by calling the DateTime.ToFileTime and DateTime.ToFileTimeUtc methods, and from DateTimeOffset values by calling the DateTimeOffset.ToFileTime method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.FromUnixTimeMilliseconds(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a Unix time expressed as the number of milliseconds that have elapsed since 1970-01-01T00:00:00Z to a DateTimeOffset value.

C#

```
public static DateTimeOffset FromUnixTimeMilliseconds (long milliseconds);
```

Parameters

`milliseconds` Int64

A Unix time, expressed as the number of milliseconds that have elapsed since 1970-01-01T00:00:00Z (January 1, 1970, at 12:00 AM UTC). For Unix times before this date, its value is negative.

Returns

DateTimeOffset

A date and time value that represents the same moment in time as the Unix time.

Exceptions

ArgumentOutOfRangeException

`milliseconds` is less than -62,135,596,800,000.

-or-

`milliseconds` is greater than 253,402,300,799,999.

Remarks

The Offset property value of the returned DateTimeOffset instance is TimeSpan.Zero, which represents Coordinated Universal Time. You can convert it to the time in a specific time zone by calling the TimeZoneInfo.ConvertTime(DateTimeOffset, TimeZoneInfo) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FromUnixTimeSeconds(Int64)
- ToUnixTimeMilliseconds()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.FromUnixTimeSeconds(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a Unix time expressed as the number of seconds that have elapsed since 1970-01-01T00:00:00Z to a DateTimeOffset value.

C#

```
public static DateTimeOffset FromUnixTimeSeconds (long seconds);
```

Parameters

seconds Int64

A Unix time, expressed as the number of seconds that have elapsed since 1970-01-01T00:00:00Z (January 1, 1970, at 12:00 AM UTC). For Unix times before this date, its value is negative.

Returns

DateTimeOffset

A date and time value that represents the same moment in time as the Unix time.

Exceptions

ArgumentOutOfRangeException

`seconds` is less than -62,135,596,800.

-or-

`seconds` is greater than 253,402,300,799.

Remarks

The Offset property value of the returned DateTimeOffset instance is TimeSpan.Zero, which represents Coordinated Universal Time. You can convert it to the time in a specific time zone by calling the TimeZoneInfo.ConvertTime(DateTimeOffset, TimeZoneInfo) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FromUnixTimeMilliseconds(Int64)
- ToUnixTimeSeconds()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current DateTimeOffset object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTimeOffset.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTimeOffset equivalent.

Overloads

Expand table

Parse(String)	Converts the specified string representation of a date, time, and offset to its DateTimeOffset equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.

Remarks

Important

Eras in the Japanese calendars are based on the emperor's reign and are therefore expected to change. For example, May 1, 2019 marked the beginning of the Reiwa era in the **JapaneseCalendar** and **JapaneseLunisolarCalendar**. Such a change of era affects all applications that use these calendars. For more information and to

determine whether your applications are affected, see **Handling a new era in the Japanese calendar in .NET**. For information on testing your applications on Windows systems to ensure their readiness for the era change, see **Prepare your application for the Japanese era change**. For features in .NET that support calendars with multiple eras and for best practices when working with calendars that support multiple eras, see **Working with eras**.

Parse(String)

Converts the specified string representation of a date, time, and offset to its DateTimeOffset equivalent.

```
C#

public static DateTimeOffset Parse (string input);
```

Parameters

`input` String

A string that contains a date and time to convert.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in `input`.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

ArgumentNullException

`input` is `null`.

FormatException

`input` does not contain a valid string representation of a date and time.

-or-

`input` contains the string representation of an offset value without a date or time.

Examples

The following example calls the Parse(String) method to parse several date and time strings. The example includes output from March 22, 2007, on a system whose culture is en-us.

C#

```
string dateString;
DateTimeOffset offsetDate;

// String with date only
dateString = "05/01/2008";
offsetDate = DateTimeOffset.Parse(dateString);
Console.WriteLine(offsetDate.ToString());

// String with time only
dateString = "11:36 PM";
offsetDate = DateTimeOffset.Parse(dateString);
Console.WriteLine(offsetDate.ToString());

// String with date and offset
dateString = "05/01/2008 +1:00";
offsetDate = DateTimeOffset.Parse(dateString);
Console.WriteLine(offsetDate.ToString());

// String with day abbreviation
dateString = "Thu May 01, 2008";
offsetDate = DateTimeOffset.Parse(dateString);
Console.WriteLine(offsetDate.ToString());
```

Remarks

Parse(String) parses a string with three elements that can appear in any order and are delimited by white space. These three elements are as shown in the following table.

Expand table

Element	Example
<Date>	"2/10/2007"
<Time>	"1:02:03 PM"
<Offset>	"-7:30"

Although each of these elements is optional, <Offset> cannot appear by itself. It must be provided together with either <Date> or <Time>. If <Date> is missing, its default value is the current day. If <Time> is missing, its default value is 12:00:00 AM. If <Date> is present but its year component consists of only two digits, it is converted to a year in the current culture's current calendar based on the value of the Calendar.TwoDigitYearMax property. If <Offset> is missing, its default value is the offset of the local time zone. <Offset> can represent either a negative or a positive offset from Coordinated Universal Time (UTC). In either case, <Offset> must include a sign symbol.

The `input` string is parsed by using the formatting information in a DateTimeFormatInfo object that is initialized for the current culture. To parse a string that contains designated formatting that does not necessarily correspond to the formatting of the current culture, use the ParseExact method and provide a format specifier.

See also

- TryParse
- ParseExact

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#
```

```
public static DateTimeOffset Parse (ReadOnlySpan<char> s,
IFormatProvider? provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

DateTimeOffset

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, IFormatProvider)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information.

C#

```
public static DateTimeOffset Parse (string input, IFormatProvider?
formatProvider);
```

Parameters

`input` String

A string that contains a date and time to convert.

`formatProvider` IFormatProvider

An object that provides culture-specific format information about `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in `input`, as specified by `formatProvider`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

ArgumentNullException

`input` is `null`.

FormatException

`input` does not contain a valid string representation of a date and time.

-or-

`input` contains the string representation of an offset value without a date or time.

Examples

The following example parses date and time strings that are formatted for the fr-fr culture and displays them using the local system's default en-us culture.

C#

```
DateTimeFormatInfo fmt = new CultureInfo("fr-fr").DateTimeFormat;
string dateString;
DateTimeOffset offsetDate;
```

```
dateString = "03-12-07";
offsetDate = DateTimeOffset.Parse(dateString, fmt);
Console.WriteLine("{0} returns {1}",
                  dateString,
                  offsetDate.ToString());

dateString = "15/09/07 08:45:00 +1:00";
offsetDate = DateTimeOffset.Parse(dateString, fmt);
Console.WriteLine("{0} returns {1}",
                  dateString,
                  offsetDate.ToString());

dateString = "mar. 1 janvier 2008 1:00:00 +1:00";
offsetDate = DateTimeOffset.Parse(dateString, fmt);
Console.WriteLine("{0} returns {1}",
                  dateString,
                  offsetDate.ToString());
// The example displays the following output to the console:
//    03-12-07 returns 12/3/2007 12:00:00 AM -08:00
//    15/09/07 08:45:00 +1:00 returns 9/15/2007 8:45:00 AM +01:00
//    mar. 1 janvier 2008 1:00:00 +1:00 returns 1/1/2008 1:00:00 AM
+01:00
```

Remarks

This method parses a string with three elements that can appear in any order and are delimited by white space. These three elements are shown in the following table.

Expand table

Element	Example
<Date>	"2/10/2007"
<Time>	"1:02:03 PM"
<Offset>	"-7:30"

Although each of these elements is optional, <Offset> cannot appear by itself. It must be provided together with either <Date> or <Time>. If <Date> is missing, its default value is the current day. If <Date> is present but its year component consists of only two digits, it is converted to a year in the `provider` parameter's current calendar based on the value of the Calendar.TwoDigitYearMax property. If <Time> is missing, its default value is 12:00:00 AM. If <Offset> is missing, its default value is the offset of the local time zone. <Offset> can represent either a negative or a positive offset from Coordinated Universal Time (UTC). In either case, <Offset> must include a sign symbol.

The format of these three elements is defined by the `formatProvider` parameter, which can be either of the following:

- A CultureInfo object that represents the culture whose formatting is used in the `input` parameter. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting used in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The positive or negative sign used in <Offset> must be either + or -. It is not defined by the PositiveSign or NegativeSign properties of the NumberFormatInfo object of the `formatProvider` parameter.

See also

- TryParse
- ParseExact

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts the specified span representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.

```
C#

public static DateTimeOffset Parse (ReadOnlySpan<char> input,
IFormatProvider? formatProvider = default,
System.Globalization.DateTimeStyles styles =
System.Globalization.DateTimeStyles.None);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a date and time to convert.

`formatProvider` IFormatProvider

An object that provides culture-specific format information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`. A typical value to specify is None.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in `input` as specified by `formatProvider` and `styles`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String, IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified culture-specific format information and formatting style.

```C#
public static DateTimeOffset Parse (string input, IFormatProvider?
formatProvider, System.Globalization.DateTimeStyles styles);
```

Parameters

`input` String

A string that contains a date and time to convert.

`formatProvider` IFormatProvider

An object that provides culture-specific format information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`. A typical value to specify is None.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in `input` as specified by `formatProvider` and `styles`.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

-or-

`styles` is not a valid DateTimeStyles value.

-or-

`styles` includes an unsupported DateTimeStyles value.

-or-

`styles` includes DateTimeStyles values that cannot be used together.

ArgumentNullException

`input` is `null`.

FormatException

`input` does not contain a valid string representation of a date and time.

-or-

`input` contains the string representation of an offset value without a date or time.

Examples

The following example illustrates the effect of passing the DateTimeStyles.AssumeLocal, DateTimeStyles.AssumeUniversal, and DateTimeStyles.AdjustToUniversal values to the `styles` parameter of the Parse(String, IFormatProvider, DateTimeStyles) method.

C#

```
string dateString;
DateTimeOffset offsetDate;

dateString = "05/01/2008 6:00:00";
// Assume time is local
offsetDate = DateTimeOffset.Parse(dateString, null,
DateTimeStyles.AssumeLocal);
Console.WriteLine(offsetDate.ToString());   // Displays 5/1/2008 6:00:00
AM -07:00

// Assume time is UTC
offsetDate = DateTimeOffset.Parse(dateString, null,
DateTimeStyles.AssumeUniversal);
Console.WriteLine(offsetDate.ToString());   // Displays 5/1/2008 6:00:00
AM +00:00

// Parse and convert to UTC
dateString = "05/01/2008 6:00:00AM +5:00";
offsetDate = DateTimeOffset.Parse(dateString, null,
DateTimeStyles.AdjustToUniversal);
Console.WriteLine(offsetDate.ToString());   // Displays 5/1/2008 1:00:00
AM +00:00
```

Remarks

This method parses a string with three elements that can appear in any order and are delimited by white space. These three elements are shown in the following table.

Expand table

Element	Example
<Date>	"2/10/2007"
<Time>	"1:02:03 PM"
<Offset>	"-7:30"

Although each of these elements is optional, <Offset> cannot appear by itself. It must be provided together with either <Date> or <Time>. If <Date> is missing, its default value is the current day. If <Date> is present but its year component consists of only two digits, it is converted to a year in the `provider` parameter's current calendar based on the value of the Calendar.TwoDigitYearMax property. If <Time> is missing, its default value is 12:00:00 AM. If <Offset> is missing, its default value is the offset of the local time zone, or TimeSpan.Zero if either the DateTimeStyles.AdjustToUniversal or DateTimeStyles.AssumeUniversal value is specified in the `styles` parameter. If <Offset> is present, it can represent either a negative or a positive offset from Coordinated Universal Time (UTC). In either case, <Offset> must include a sign symbol.

The format of these three elements is defined by the `formatProvider` parameter, which can be either of the following:

- A CultureInfo object that represents the culture whose formatting is used in the `input` parameter. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting used in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

Each element can also be enclosed by leading or trailing white space, and the <Date> and <Time> elements can include inner white space (such as 6: 00:00). Only the <Offset> component cannot include inner white space.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The positive or negative sign used in <Offset> must be either + or -. It is not defined by the PositiveSign or NegativeSign properties of the NumberFormatInfo object returned by the `formatProvider` parameter.

The following table shows the members of the System.Globalization.DateTimeStyles enumeration that are supported.

Expand table

DateTimeStyles member	Description
AdjustToUniversal	Parses the string represented by `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AllowInnerWhite	Although valid, this value is ignored. Inner white space is permitted in the <Date> and <Time> elements.
AllowLeadingWhite	Although valid, this value is ignored. Leading white space is permitted in front of each component in the parsed string.
AllowTrailingWhite	Although valid, this value is ignored. Trailing white space is permitted in front of each component in the parsed string.
AllowWhiteSpaces	This is the default behavior. It cannot be overridden by supplying a more restrictive DateTimeStyles enumeration value, such as DateTimeStyles.None.
AssumeLocal	Indicates that, if the `input` parameter lacks an <Offset> element, the offset of the local time zone should be provided. This is the default behavior of the Parse method.
AssumeUniversal	Indicates that, if the `input` parameter lacks an <Offset> element, the UTC offset (00:00) should be provided.
None	Although valid, this value is ignored and has no effect.
RoundtripKind	Because the DateTimeOffset structure does not include a `Kind` property, this value has no effect.

Only the DateTimeStyles.NoCurrentDateDefault value is not supported. An ArgumentException is thrown if this value is included in the `styles` parameter.

See also

- TryParse
- ParseExact

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTimeOffset equivalent. The format of the string representation must match a specified format exactly.

Overloads

Expand table

ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.
ParseExact(String, String, IFormatProvider)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a character span that represents a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts a character span that contains the string representation of a date and time to its DateTimeOffset equivalent using the specified

formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.

ParseExact(String, String[], IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.

```
C#

public static DateTimeOffset ParseExact (string input, string[] formats,
IFormatProvider? formatProvider, System.Globalization.DateTimeStyles
styles);
```

Parameters

`input` String

A string that contains a date and time to convert.

`formats` String[]

An array of format specifiers that define the expected formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in the `input` parameter, as specified by the `formats`, `formatProvider`, and `styles` parameters.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

-or-

`styles` includes an unsupported value.

-or-

The `styles` parameter contains DateTimeStyles values that cannot be used together.

ArgumentNullException

`input` is `null`.

FormatException

`input` is an empty string ("").

-or-

`input` does not contain a valid string representation of a date and time.

-or-

No element of `formats` contains a valid format specifier.

-or-

The hour component and the AM/PM designator in `input` do not agree.

Examples

The following example defines multiple input formats for the string representation of a date and time and offset value, and then passes the string that is entered by the user to the DateTimeOffset.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method.

C#

```
TextReader conIn = Console.In;
TextWriter conOut = Console.Out;
int tries = 0;
string input = String.Empty;
string[] formats = new string[] {@"@M/dd/yyyy HH:m zzz", @"MM/dd/yyyy
HH:m zzz",
```

```
                                    @"M/d/yyyy HH:m zzz", @"MM/d/yyyy HH:m
zzz",
                                    @"M/dd/yy HH:m zzz", @"MM/dd/yy HH:m
zzz",
                                    @"M/d/yy HH:m zzz", @"MM/d/yy HH:m zzz",
                                    @"M/dd/yyyy H:m zzz", @"MM/dd/yyyy H:m
zzz",
                                    @"M/d/yyyy H:m zzz", @"MM/d/yyyy H:m
zzz",
                                    @"M/dd/yy H:m zzz", @"MM/dd/yy H:m zzz",
                                    @"M/d/yy H:m zzz", @"MM/d/yy H:m zzz",
                                    @"M/dd/yyyy HH:mm zzz", @"MM/dd/yyyy
HH:mm zzz",
                                    @"M/d/yyyy HH:mm zzz", @"MM/d/yyyy HH:mm
zzz",
                                    @"M/dd/yy HH:mm zzz", @"MM/dd/yy HH:mm
zzz",
                                    @"M/d/yy HH:mm zzz", @"MM/d/yy HH:mm
zzz",
                                    @"M/dd/yyyy H:mm zzz", @"MM/dd/yyyy H:mm
zzz",
                                    @"M/d/yyyy H:mm zzz", @"MM/d/yyyy H:mm
zzz",
                                    @"M/dd/yy H:mm zzz", @"MM/dd/yy H:mm
zzz",
                                    @"M/d/yy H:mm zzz", @"MM/d/yy H:mm
zzz"};
IFormatProvider provider = CultureInfo.InvariantCulture.DateTimeFormat;
DateTimeOffset result = new DateTimeOffset();

do {
   conOut.WriteLine("Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-
HH:MM),");
   conOut.Write("Then press Enter: ");
   input = conIn.ReadLine();
   conOut.WriteLine();
   try
   {
      result = DateTimeOffset.ParseExact(input, formats, provider,

DateTimeStyles.AllowWhiteSpaces);
      break;
   }
   catch (FormatException)
   {
      Console.WriteLine("Unable to parse {0}.", input);
      tries++;
   }
} while (tries < 3);
if (tries >= 3)
   Console.WriteLine("Exiting application without parsing {0}", input);
else
   Console.WriteLine("{0} was converted to {1}", input,
result.ToString());
// Some successful sample interactions with the user might appear as
```

```
follows:
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/08/2007 6:54 -6:00
//
//    12/08/2007 6:54 -6:00 was converted to 12/8/2007 6:54:00 AM -06:00
//
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/8/2007 06:54 -06:00
//
//    12/8/2007 06:54 -06:00 was converted to 12/8/2007 6:54:00 AM -06:00
//
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/5/07 6:54 -6:00
//
//    12/5/07 6:54 -6:00 was converted to 12/5/2007 6:54:00 AM -06:00
```

Remarks

The DateTimeOffset.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method parses the string representation of a date that matches any one of the patterns assigned to the `formats` parameter. If the `input` string does not match any one of these patterns with any of the variations defined by the `styles` parameter, the method throws a FormatException. Aside from comparing `input` to multiple formatting patterns, this overload behaves identically to the DateTimeOffset.ParseExact(String, String, IFormatProvider, DateTimeStyles) method.

The `formats` parameter is a string array whose elements contain either a single standard format specifier or one or more custom format specifiers that define the possible pattern of the `input` parameter. When the method is called, `input` must match one of these patterns. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If the matched element in `formats` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the method throws a FormatException.

> **ⓘ Important**
>
> Using the `formats` parameter of this overload to specify multiple formats can help reduce the frustration many users experience when they enter dates and times. In particular, the ability to define multiple input patterns enables an application to handle date and time representations that can either include or

lack leading zeros in months, days, hours, minutes, and seconds. The example provides an illustration of this.

If the matched element in `formats` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If the matched element in `formats` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If the matched element in `formats` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise format of `input`, if the matching element of `formats` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is permitted in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default value.
AssumeUniversal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by `format`. Extra white space can appear between date and time components and within individual components (except the offset), and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by `formats`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by `formats`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by `formats`. All extra white-space characters not specified in the matched element in `formats` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in a particular element of `formats` for a match to occur. This is the default behavior.
RoundtripKind	Has no effect because the DateTimeOffset structure does not include a `Kind` property.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact

- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(String, String, IFormatProvider, DateTimeStyles)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.

C#

```
public static DateTimeOffset ParseExact (string input, string format,
IFormatProvider? formatProvider, System.Globalization.DateTimeStyles
styles);
```

Parameters

`input` String

A string that contains a date and time to convert.

`format` String

A format specifier that defines the expected format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in the `input` parameter, as specified by the `format`, `formatProvider`, and `styles` parameters.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

-or-

The `styles` parameter includes an unsupported value.

-or-

The `styles` parameter contains DateTimeStyles values that cannot be used together.

ArgumentNullException

`input` is `null`.

-or-

`format` is `null`.

FormatException

`input` is an empty string ("").

-or-

`input` does not contain a valid string representation of a date and time.

-or-

`format` is an empty string.

-or-

The hour component and the AM/PM designator in `input` do not agree.

Examples

The following example uses the DateTimeOffset.ParseExact(String, String, IFormatProvider, DateTimeStyles) method with standard and custom format specifiers, the invariant culture, and various DateTimeStyles values to parse several date and time strings.

C#

```
string dateString, format;
DateTimeOffset result;
CultureInfo provider = CultureInfo.InvariantCulture;

// Parse date-only value with invariant culture and assume time is UTC.
dateString = "06/15/2008";
format = "d";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider,
                                      DateTimeStyles.AssumeUniversal);
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("'{0}' is not in the correct format.", dateString);
}

// Parse date-only value with leading white space.
// Should throw a FormatException because only trailing white space is
// specified in method call.
dateString = " 06/15/2008";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider,
                                      DateTimeStyles.AllowTrailingWhite);
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("'{0}' is not in the correct format.", dateString);
}

// Parse date and time value, and allow all white space.
dateString = " 06/15/   2008  15:15    -05:00";
format = "MM/dd/yyyy H:mm zzz";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider,
                                      DateTimeStyles.AllowWhiteSpaces);
   Console.WriteLine("'{0}' converts to {1}.", dateString,
```

```
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("'{0}' is not in the correct format.", dateString);
}

// Parse date and time and convert to UTC.
dateString = "  06/15/2008 15:15:30 -05:00";
format = "MM/dd/yyyy H:mm:ss zzz";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider,
                                      DateTimeStyles.AllowWhiteSpaces |
                                      DateTimeStyles.AdjustToUniversal);
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("'{0}' is not in the correct format.", dateString);
}
// The example displays the following output:
//    '06/15/2008' converts to 6/15/2008 12:00:00 AM +00:00.
//    ' 06/15/2008' is not in the correct format.
//    ' 06/15/   2008  15:15    -05:00' converts to 6/15/2008 3:15:00 PM
-05:00.
//    '  06/15/2008 15:15:30 -05:00' converts to 6/15/2008 8:15:30 PM
+00:00.
```

The following example uses a variety of DateTimeStyles values to parse an array of strings that are expected to conform to ISO 8601. As the output from the example shows, strings that are in the proper format fail to parse if:

- they contain white space, and an appropriate DateTimeStyles flag (such as DateTimeStyles.AllowWhiteSpaces has not been supplied in the method call.
- they contain date and time elements that are out of range.

Strings that do not specify a UTC offset are assumed to have the offset of the local time zone (in this case, -07:00) unless the DateTimeStyles.AssumeUniversal flag is supplied in the method call. In that case, they are assumed to be Universal Coordinated Time.

C#

```
using System;
using System.Globalization;

public class Example
{
```

```
   public static void Main()
   {
      string[] dateStrings = { "2018-08-18T12:45:16.0000000Z",
                               "2018/08/18T12:45:16.0000000Z",
                               "2018-18-08T12:45:16.0000000Z",
                               "2018-08-18T12:45:16.0000000",
                               " 2018-08-18T12:45:16.0000000Z ",
                               "2018-08-18T12:45:16.0000000+02:00",
                               "2018-08-18T12:45:16.0000000-07:00" };

      ParseWithISO8601(dateStrings, DateTimeStyles.None);
      Console.WriteLine("\n-----\n");
      ParseWithISO8601(dateStrings, DateTimeStyles.AllowWhiteSpaces);
      Console.WriteLine("\n-----\n");
      ParseWithISO8601(dateStrings, DateTimeStyles.AdjustToUniversal);
      Console.WriteLine("\n-----\n");
      ParseWithISO8601(dateStrings, DateTimeStyles.AssumeLocal);
      Console.WriteLine("\n-----\n");
      ParseWithISO8601(dateStrings, DateTimeStyles.AssumeUniversal);   }

   private static void ParseWithISO8601(string[] dateStrings,
DateTimeStyles styles)
   {
      Console.WriteLine($"Parsing with {styles}:");
      foreach (var dateString in dateStrings)
      {
         try {
            var date = DateTimeOffset.ParseExact(dateString, "O", null,
styles);
            Console.WriteLine($"   {dateString,-35} --> {date:yyyy-MM-dd
HH:mm:ss.FF zzz}");
         }
         catch (FormatException)
         {
            Console.WriteLine($"   FormatException: Unable to convert
'{dateString}'");
         }
      }
   }
}
// The example displays the following output:
//      Parsing with None:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert
'2018/08/18T12:45:16.0000000Z'
//         FormatException: Unable to convert '2018-18-
08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//         FormatException: Unable to convert ' 2018-08-
18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
```

```
-07:00
//
//      -----
//
//      Parsing with AllowWhiteSpaces:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert
'2018/08/18T12:45:16.0000000Z'
//         FormatException: Unable to convert '2018-18-
08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//         2018-08-18T12:45:16.0000000Z       --> 2018-08-18 12:45:16
+00:00
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
//
//      -----
//
//      Parsing with AdjustToUniversal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert
'2018/08/18T12:45:16.0000000Z'
//         FormatException: Unable to convert '2018-18-
08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 19:45:16
+00:00
//         FormatException: Unable to convert ' 2018-08-
18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 10:45:16
+00:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 19:45:16
+00:00
//
//      -----
//
//      Parsing with AssumeLocal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert
'2018/08/18T12:45:16.0000000Z'
//         FormatException: Unable to convert '2018-18-
08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//         FormatException: Unable to convert ' 2018-08-
18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
```

```
//
//      -----
//
//      Parsing with AssumeUniversal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert
'2018/08/18T12:45:16.0000000Z'
//         FormatException: Unable to convert '2018-18-
08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
+00:00
//         FormatException: Unable to convert ' 2018-08-
18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
```

Remarks

The DateTimeOffset.ParseExact(String, String, IFormatProvider, DateTimeStyles) method parses the string representation of a date, which must be in the format defined by the `format` parameter. It also requires that the <Date>, <Time>, and <Offset> elements of the string representation of a date and time appear in the order specified by `format`. If the `input` string does not match the pattern of the `format` parameter, with any variations defined by the `styles` parameter, the method throws a FormatException. In contrast, the DateTimeOffset.Parse(String, IFormatProvider, DateTimeStyles) method parses the string representation of a date in any one of the formats recognized by the format provider's DateTimeFormatInfo object. Parse also allows the <Date>, <Time>, and <Offset> elements of the string representation of a date and time to appear in any order.

The `format` parameter is a string that contains either a single standard format specifier or one or more custom format specifiers that define the required pattern of the `input` parameter. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If `format` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the method throws a FormatException.

If `format` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If `format` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is

assigned the current date on the local system. If `format` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise format of `input`, if `format` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is allowed in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default value.
AssumeUniversal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).

`DateTimeStyles` member	Behavior
AllowInnerWhite	Allows `input` to include inner white space not specified by `format`. Extra white space can appear between date and time components and within individual components, and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by `format`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by `format`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by `format`. All extra white-space characters not specified in `format` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in `format`. This is the default behavior.
RoundtripKind	Has no effect because the DateTimeOffset structure does not include a `Kind` property.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(String, String, IFormatProvider)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static DateTimeOffset ParseExact (string input, string format,
IFormatProvider? formatProvider);
```

Parameters

`input` String

A string that contains a date and time to convert.

`format` String

A format specifier that defines the expected format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in `input` as specified by `format` and `formatProvider`.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours.

ArgumentNullException

`input` is `null`.

-or-

`format` is `null`.

FormatException

`input` is an empty string ("").

-or-

`input` does not contain a valid string representation of a date and time.

-or-

`format` is an empty string.

-or-

The hour component and the AM/PM designator in `input` do not agree.

Examples

The following example uses the DateTimeOffset.ParseExact(String, String, IFormatProvider) method with standard and custom format specifiers and the invariant culture to parse several date and time strings.

C#

```
string dateString, format;
DateTimeOffset result;
CultureInfo provider = CultureInfo.InvariantCulture;

// Parse date-only value with invariant culture.
dateString = "06/15/2008";
format = "d";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider);
   Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("{0} is not in the correct format.", dateString);
}
```

```
// Parse date-only value without leading zero in month using "d" format.
// Should throw a FormatException because standard short date pattern of
// invariant culture requires two-digit month.
dateString = "6/15/2008";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider);
   Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("{0} is not in the correct format.", dateString);
}

// Parse date and time with custom specifier.
dateString = "Sun 15 Jun 2008 8:30 AM -06:00";
format = "ddd dd MMM yyyy h:mm tt zzz";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider);
   Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("{0} is not in the correct format.", dateString);
}

// Parse date and time with offset without offset//s minutes.
// Should throw a FormatException because "zzz" specifier requires
leading
// zero in hours.
dateString = "Sun 15 Jun 2008 8:30 AM -06";
try
{
   result = DateTimeOffset.ParseExact(dateString, format, provider);
   Console.WriteLine("{0} converts to {1}.", dateString,
result.ToString());
}
catch (FormatException)
{
   Console.WriteLine("{0} is not in the correct format.", dateString);
}
// The example displays the following output:
//    06/15/2008 converts to 6/15/2008 12:00:00 AM -07:00.
//    6/15/2008 is not in the correct format.
//    Sun 15 Jun 2008 8:30 AM -06:00 converts to 6/15/2008 8:30:00 AM
-06:00.
//    Sun 15 Jun 2008 8:30 AM -06 is not in the correct format.
```

The following example parses an array of strings that are expected to conform to ISO 8601. As the output from the example shows, strings with leading or trailing spaces

fail to parse successfully, as do strings with date and time elements that are out of range.

C#

```
using System;

public class Example2
{
   public static void Main()
   {
      string[] dateStrings = { "2018-08-18T12:45:16.0000000Z",
                               "2018/08/18T12:45:16.0000000Z",
                               "2018-18-08T12:45:16.0000000Z",
                               " 2018-08-18T12:45:16.0000000Z ",
                               "2018-08-18T12:45:16.0000000+02:00",
                               "2018-08-18T12:45:16.0000000-07:00" };

      foreach (var dateString in dateStrings)
      {
         try {
            var date = DateTimeOffset.ParseExact(dateString, "O", null);
            Console.WriteLine($"{dateString,-35} --> {date:yyyy-MM-dd
HH:mm:ss.FF zzz}");
         }
         catch (FormatException)
         {
            Console.WriteLine($"FormatException: Unable to convert
'{dateString}'");
         }
      }
   }
}
// The example displays the following output:
//      2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//      FormatException: Unable to convert '2018/08/18T12:45:16.0000000Z'
//      FormatException: Unable to convert '2018-18-08T12:45:16.0000000Z'
//      FormatException: Unable to convert ' 2018-08-18T12:45:16.0000000Z
'
//      2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//      2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
```

Remarks

The ParseExact(String, String, IFormatProvider) method parses the string representation of a date, which must be in the format defined by the `format` parameter. It also requires that the <Date>, <Time>, and <Offset> elements of the

string representation of a date and time appear in the order specified by `format`. If the `input` string does not match this `format` parameter, the method throws a FormatException. In contrast, the DateTimeOffset.Parse(String, IFormatProvider) method parses the string representation of a date in any one of the formats recognized by the format provider's DateTimeFormatInfo object. Parse also allows the <Date>, <Time>, and <Offset> elements of the string representation of a date and time to appear in any order.

The `format` parameter is a string that contains either a single standard format specifier or one or more custom format specifiers that define the required format of the `input` parameter. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If `format` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the method throws a FormatException.

If `format` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If `format` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If `format` does not require that `input` contain an offset, the resulting DateTimeOffset object is assigned the time zone offset of the local system.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter, as is the precise format of `input` if `format` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

Notes to Callers

In the .NET Framework 4, the ParseExact method throws a FormatException if the string to be parsed contains an hour component and an AM/PM designator that are

not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- TryParseExact
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts a character span that represents a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.

C#

```
public static DateTimeOffset ParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, IFormatProvider? formatProvider,
System.Globalization.DateTimeStyles styles =
System.Globalization.DateTimeStyles.None);
```

Parameters

`input` ReadOnlySpan<Char>

A character span that represents a date and time.

`format` ReadOnlySpan<Char>

A character span that contains a format specifier that defines the expected format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in the `input` parameter, as specified by the `format`, `formatProvider`, and `styles` parameters.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours. -or- The `styles` parameter includes an unsupported value. -or- The `styles` parameter contains DateTimeStyles values that cannot be used together.

FormatException

`input` is an empty character span. -or- `input` does not contain a valid string representation of a date and time. -or- `format` is an empty character span. -or- The hour component and the AM/PM designator in `input` do not agree.

Remarks

This method parses a character span that represents a date, which must be in the format defined by the `format` parameter. It also requires that the <Date>, <Time>, and <Offset> elements of the string representation of a date and time appear in the order specified by `format`. If `input` does not match the `format` pattern, the method

throws a FormatException. In contrast, the DateTimeOffset.Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles) method parses the string representation of a date in any one of the formats recognized by the format provider's DateTimeFormatInfo object. Parse also allows the <Date>, <Time>, and <Offset> elements of the string representation of a date and time to appear in any order.

The `format` parameter is a character span that contains either a single-character standard format specifier or one or more custom format specifiers that define the required format of the `input` parameter. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If `format` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the method throws a FormatException.

If `format` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If `format` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If `format` does not require that `input` contain an offset, the resulting DateTimeOffset object is assigned the time zone offset of the local system.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter, as is the precise format of `input` if `format` is a standard format specifier. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by its DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)

Converts a character span that contains the string representation of a date and time to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.

C#

```
public static DateTimeOffset ParseExact (ReadOnlySpan<char> input,
string[] formats, IFormatProvider? formatProvider,
System.Globalization.DateTimeStyles styles =
System.Globalization.DateTimeStyles.None);
```

Parameters

`input` ReadOnlySpan<Char>

A character span that contains a date and time to convert.

`formats` String[]

An array of format specifiers that define the expected formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

Returns

DateTimeOffset

An object that is equivalent to the date and time that is contained in the `input` parameter, as specified by the `formats`, `formatProvider`, and `styles` parameters.

Exceptions

ArgumentException

The offset is greater than 14 hours or less than -14 hours. -or- `styles` includes an unsupported value. -or- The `styles` parameter contains DateTimeStyles values that cannot be used together.

FormatException

`input` is an empty character span. -or- `input` does not contain a valid string representation of a date and time. -or- No element of `formats` contains a valid format specifier. -or- The hour component and the AM/PM designator in `input` do not agree.

Remarks

This method parses a character span representing a date that matches any one of the patterns assigned to the `formats` parameter. If `input` does not match any one of these patterns with any of the variations defined by the `styles` parameter, the method throws a FormatException. Aside from comparing `input` to multiple formatting patterns, this overload behaves identically to the DateTimeOffset.ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles) method.

The `formats` parameter is a string array whose elements contain either a single standard format specifier or one or more custom format specifiers that define the possible pattern of the `input` parameter. When the method is called, `input` must match one of these patterns. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If the matched element in `formats` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the method throws a FormatException.

> **ⓘ Important**
>
> Using the `formats` parameter of this overload to specify multiple formats can help reduce the frustration many users experience when they enter dates and times. In particular, the ability to define multiple input patterns enables an application to handle date and time representations that can either include or lack leading zeros in months, days, hours, minutes, and seconds.

If the matched element in `formats` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If the matched element in `formats` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If the matched element in `formats` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise format of `input`, if the matching element of `formats` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is permitted in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.

`DateTimeStyles` member	Behavior
AssumeLocal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default value.
AssumeUniversal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by `format`. Extra white space can appear between date and time components and within individual components (except the offset), and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by `formats`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by `formats`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by `formats`. All extra white-space characters not specified in the matched element in `formats` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in a particular element of `formats` for a match to occur. This is the default behavior.
RoundtripKind	Has no effect because the DateTimeOffset structure does not include a `Kind` property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.Subtract Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts a specified time or duration from the current DateTimeOffset object.

Overloads

Expand table

Subtract(DateTimeOffset)	Subtracts a DateTimeOffset value that represents a specific date and time from the current DateTimeOffset object.
Subtract(TimeSpan)	Subtracts a specified time interval from the current DateTimeOffset object.

Subtract(DateTimeOffset)

Subtracts a DateTimeOffset value that represents a specific date and time from the current DateTimeOffset object.

C#

```
public TimeSpan Subtract (DateTimeOffset value);
```

Parameters

`value` DateTimeOffset

An object that represents the value to subtract.

Returns

TimeSpan

An object that specifies the interval between the two DateTimeOffset objects.

Examples

The following example illustrates subtraction that uses the Subtract(DateTimeOffset) method.

C#

```
DateTimeOffset firstDate = new DateTimeOffset(2018, 10, 25, 18, 0, 0,
                                              new TimeSpan(-7, 0, 0));
DateTimeOffset secondDate = new DateTimeOffset(2018, 10, 25, 18, 0, 0,
                                               new TimeSpan(-5, 0, 0));
DateTimeOffset thirdDate = new DateTimeOffset(2018, 9, 28, 9, 0, 0,
                                              new TimeSpan(-7, 0, 0));
TimeSpan difference;

difference = firstDate.Subtract(secondDate);
Console.WriteLine($"({firstDate}) - ({secondDate}): {difference.Days}
days, {difference.Hours}:{difference.Minutes:d2}");

difference = firstDate.Subtract(thirdDate);
Console.WriteLine($"({firstDate}) - ({thirdDate}): {difference.Days}
days, {difference.Hours}:{difference.Minutes:d2}");

// The example produces the following output:
//    (10/25/2018 6:00:00 PM -07:00) - (10/25/2018 6:00:00 PM -05:00): 0
days, 2:00
//    (10/25/2018 6:00:00 PM -07:00) - (9/28/2018 9:00:00 AM -07:00): 27
days, 9:00
```

Remarks

This method converts both DateTimeOffset objects to Coordinated Universal Time (UTC) before calculating the time interval that separates them. This removes any effect that different offsets from UTC may have on the comparison.

> **Note**
>
> For languages that support custom operators, you can also perform date and time subtraction by using the subtraction operator. For details, see the **Subtraction** method.

Because neither the current DateTimeOffset object nor the `value` parameter represent the date and time in a specific time zone, the Subtract(DateTimeOffset) method does not consider a particular time zone's adjustment rules when it subtracts dates and times.

See also

- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Subtract(TimeSpan)

Subtracts a specified time interval from the current DateTimeOffset object.

C#

```
public DateTimeOffset Subtract (TimeSpan value);
```

Parameters

`value` TimeSpan

The time interval to subtract.

Returns

DateTimeOffset

An object that is equal to the date and time represented by the current DateTimeOffset object, minus the time interval represented by `value`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Examples

The following example illustrates subtraction that uses the Subtract method.

C#

```
DateTimeOffset offsetDate = new DateTimeOffset(2007, 12, 3, 11, 30, 0,
                                new TimeSpan(-8, 0, 0));
TimeSpan duration = new TimeSpan(7, 18, 0, 0);
Console.WriteLine(offsetDate.Subtract(duration).ToString());  // Displays
11/25/2007 5:30:00 PM -08:00
```

Remarks

You can use the Subtract method to subtract more than one kind of time interval (days, hours, minutes, seconds, or milliseconds) in a single operation. Its behavior is identical to the Subtraction(DateTimeOffset, TimeSpan) method, which defines the subtraction operator. The DateTimeOffset structure also supports specialized addition methods (such as AddDays, AddHours, and AddMinutes) that allow you to perform subtraction by assigning a negative value to the method parameter.

> **Note**
>
> This method returns a new **DateTimeOffset** object. It does not modify the value of the current object by subtracting `value` from its date and time.

The Subtract method does not affect the value of the DateTimeOffset object's Offset property. The returned DateTimeOffset object has the same offset as the original object.

Because the current DateTimeOffset object does not represent the date and time in a specific time zone, the Subtract(TimeSpan) method does not consider a particular time zone's adjustment rules when it performs the subtraction.

See also

- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToFileTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTimeOffset object to a Windows file time.

C#

```
public long ToFileTime ();
```

Returns

Int64

The value of the current DateTimeOffset object, expressed as a Windows file time.

Exceptions

ArgumentOutOfRangeException

The resulting file time would represent a date and time before midnight on January 1, 1601 C.E. Coordinated Universal Time (UTC).

Remarks

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC). Windows uses a file time to record when an application creates, accesses, or writes to a file.

The ToFileTime method converts the current DateTimeOffset object's date and time to UTC before it performs the conversion. In other words, calling the ToFileTime method is equivalent to the following method call:

C#

```
this.ToUtcDateTime().ToFileTime();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FromFileTime(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToLocalTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current DateTimeOffset object to a DateTimeOffset object that represents the local time.

C#

```
public DateTimeOffset ToLocalTime ();
```

Returns

DateTimeOffset

An object that represents the date and time of the current DateTimeOffset object converted to local time.

Examples

The following example uses the ToLocalTime method to convert a DateTimeOffset value to local time in the Pacific Standard Time zone. It also illustrates the method's support for the local time zone's adjustment rules.

C#

```
// Local time changes on 3/11/2007 at 2:00 AM
DateTimeOffset originalTime, localTime;

originalTime = new DateTimeOffset(2007, 3, 11, 3, 0, 0,
                                  new TimeSpan(-6, 0, 0));
localTime = originalTime.ToLocalTime();
Console.WriteLine("Converted {0} to {1}.", originalTime.ToString(),
                                           localTime.ToString());

originalTime = new DateTimeOffset(2007, 3, 11, 4, 0, 0,
                                  new TimeSpan(-6, 0, 0));
localTime = originalTime.ToLocalTime();
Console.WriteLine("Converted {0} to {1}.", originalTime.ToString(),
```

```
                                            localTime.ToString());

// Define a summer UTC time
originalTime = new DateTimeOffset(2007, 6, 15, 8, 0, 0,
                                  TimeSpan.Zero);
localTime = originalTime.ToLocalTime();
Console.WriteLine("Converted {0} to {1}.", originalTime.ToString(),
                                            localTime.ToString());

// Define a winter time
originalTime = new DateTimeOffset(2007, 11, 30, 14, 0, 0,
                                  new TimeSpan(3, 0, 0));
localTime = originalTime.ToLocalTime();
Console.WriteLine("Converted {0} to {1}.", originalTime.ToString(),
                                            localTime.ToString());
// The example produces the following output:
//    Converted 3/11/2007 3:00:00 AM -06:00 to 3/11/2007 1:00:00 AM -08:00.
//    Converted 3/11/2007 4:00:00 AM -06:00 to 3/11/2007 3:00:00 AM -07:00.
//    Converted 6/15/2007 8:00:00 AM +00:00 to 6/15/2007 1:00:00 AM -07:00.
//    Converted 11/30/2007 2:00:00 PM +03:00 to 11/30/2007 3:00:00 AM
-08:00.
```

Remarks

In performing the conversion to local time, the method first converts the current DateTimeOffset object's date and time to Coordinated Universal Time (UTC) by subtracting the offset from the time. It then converts the UTC date and time to local time by adding the local time zone offset. In doing this, it takes account of any adjustment rules for the local time zone.

Both the value of the current DateTimeOffset object and the value of the DateTimeOffset object returned by the method call represent the same point in time. That is, if both are passed to the DateTimeOffset.Equals(DateTimeOffset, DateTimeOffset) method, the method will return `true`.

If the conversion causes a time that is out of range of the DateTimeOffset type, the ToLocalTime method returns a DateTimeOffset object that has the date and time set to either MaxValue or MinValue and the offset set to the local time zone offset.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToUniversalTime()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToOffset(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTimeOffset object to the date and time specified by an offset value.

```
C#

public DateTimeOffset ToOffset (TimeSpan offset);
```

Parameters

`offset` TimeSpan

The offset to convert the DateTimeOffset value to.

Returns

DateTimeOffset

An object that is equal to the original DateTimeOffset object (that is, their ToUniversalTime() methods return identical points in time) but whose Offset property is set to `offset`.

Exceptions

ArgumentException

The resulting DateTimeOffset object has a DateTime value earlier than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset object has a DateTime value later than DateTimeOffset.MaxValue.

ArgumentOutOfRangeException

`offset` is less than -14 hours.

-or-

`offset` is greater than 14 hours.

Examples

The following example illustrates how to use the ToOffset method to convert a DateTimeOffset object to a DateTimeOffset object with a different offset.

C#

```
using System;

public class DateTimeOffsetConversion
{
   private static DateTimeOffset sourceTime;

   public static void Main()
   {
      DateTimeOffset targetTime;
      sourceTime = new DateTimeOffset(2007, 9, 1, 9, 30, 0,
                                      new TimeSpan(-5, 0, 0));

      // Convert to same time (return sourceTime unchanged)
      targetTime = sourceTime.ToOffset(new TimeSpan(-5, 0, 0));
      ShowDateAndTimeInfo(targetTime);

      // Convert to UTC (0 offset)
      targetTime = sourceTime.ToOffset(TimeSpan.Zero);
      ShowDateAndTimeInfo(targetTime);

      // Convert to 8 hours behind UTC
      targetTime = sourceTime.ToOffset(new TimeSpan(-8, 0, 0));
      ShowDateAndTimeInfo(targetTime);

      // Convert to 3 hours ahead of UTC
      targetTime = sourceTime.ToOffset(new TimeSpan(3, 0, 0));
      ShowDateAndTimeInfo(targetTime);
   }

   private static void ShowDateAndTimeInfo(DateTimeOffset newTime)
   {
      Console.WriteLine("{0} converts to {1}", sourceTime, newTime);
      Console.WriteLine("{0} and {1} are equal: {2}",
                        sourceTime, newTime, sourceTime.Equals(newTime));
      Console.WriteLine("{0} and {1} are identical: {2}",
                        sourceTime, newTime,
                        sourceTime.EqualsExact(newTime));
```

```
        Console.WriteLine();
    }
}
//
// The example displays the following output:
//    9/1/2007 9:30:00 AM -05:00 converts to 9/1/2007 9:30:00 AM -05:00
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 9:30:00 AM -05:00 are equal:
True
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 9:30:00 AM -05:00 are
identical: True
//
//    9/1/2007 9:30:00 AM -05:00 converts to 9/1/2007 2:30:00 PM +00:00
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 2:30:00 PM +00:00 are equal:
True
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 2:30:00 PM +00:00 are
identical: False
//
//    9/1/2007 9:30:00 AM -05:00 converts to 9/1/2007 6:30:00 AM -08:00
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 6:30:00 AM -08:00 are equal:
True
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 6:30:00 AM -08:00 are
identical: False
//
//    9/1/2007 9:30:00 AM -05:00 converts to 9/1/2007 5:30:00 PM +03:00
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 5:30:00 PM +03:00 are equal:
True
//    9/1/2007 9:30:00 AM -05:00 and 9/1/2007 5:30:00 PM +03:00 are
identical: False
```

Remarks

The ToOffset method is an alternative to calling the TimeZoneInfo.ConvertTime(DateTimeOffset, TimeZoneInfo) method. It can be useful for performing simple conversions from one time zone to another when the time zones' offsets from Coordinated Universal Time (UTC) are known. However, because neither the original DateTimeOffset object nor the new DateTimeOffset object returned by the method call are unambiguously related to a particular time zone, the method does not apply any time zone adjustment rules in the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current DateTimeOffset object to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the value of the current DateTimeOffset object to its equivalent string representation.
ToString(IFormatProvider)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified culture-specific formatting information.
ToString(String)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the value of the current DateTimeOffset object to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

A string representation of a DateTimeOffset object that includes the offset appended at the end of the string.

Exceptions

ArgumentOutOfRangeException
The date and time is outside the range of dates supported by the calendar used by the current culture.

Examples

The following example illustrates calls to the ToString() method and displays its output on a system whose current culture is en-us.

C#

```
DateTimeOffset thisDate;

// Show output for UTC time
thisDate = DateTimeOffset.UtcNow;
Console.WriteLine(thisDate.ToString());  // Displays 3/28/2007 7:13:50 PM
+00:00

// Show output for local time
thisDate = DateTimeOffset.Now;
Console.WriteLine(thisDate.ToString());  // Displays 3/28/2007 12:13:50
PM -07:00

// Show output for arbitrary time offset
thisDate = thisDate.ToOffset(new TimeSpan(-5, 0, 0));
Console.WriteLine(thisDate.ToString());  // Displays 3/28/2007 2:13:50 PM
-05:00
```

Remarks

The return value of this method is identical to that of the DateTime.ToString() method, except that it includes a space followed by the offset appended at the end of the string. In other words, it formats output using the short date pattern, the long time pattern, and the `zzz` custom format string, with each element separated from the previous element by a space. For example, if DateTime.ToString() returns a value of 1/12/2008 6:15:50 PM, ToString() returns a value of 1/12/2008 6:15:50 PM -08:00 for a time that is eight hours behind Coordinated Universal Time (UTC).

This method uses formatting information derived from the current culture. For more information, see CurrentCulture. Other overloads of the ToString method enable you

to specify the culture whose formatting to use, and to define the output pattern of the DateTimeOffset value.

Notes to Callers

The ToString() method returns the string representation of the date and time in the calendar used by the current culture. If the value of the current DateTimeOffset instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the HijriCalendar class when the current culture is Arabic (Syria).

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
   public static void Main()
   {
      DateTimeOffset date1 = new DateTimeOffset(new DateTime(550, 1, 1),
                                                TimeSpan.Zero);
      CultureInfo dft;
      CultureInfo arSY = new CultureInfo("ar-SY");
      arSY.DateTimeFormat.Calendar = new HijriCalendar();

      // Change current culture to ar-SY.
      dft = Thread.CurrentThread.CurrentCulture;
      Thread.CurrentThread.CurrentCulture = arSY;

      // Display the date using the current culture's calendar.
      try {
         Console.WriteLine(date1.ToString());
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("{0} is earlier than {1} or later than {2}",
                           date1.ToString("d",
CultureInfo.InvariantCulture),

arSY.DateTimeFormat.Calendar.MinSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture),

arSY.DateTimeFormat.Calendar.MaxSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture));
      }

      // Restore the default culture.
      Thread.CurrentThread.CurrentCulture = dft;
```

```
    }
}
// The example displays the following output:
//    01/01/0550 is earlier than 07/18/0622 or later than 12/31/9999
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified culture-specific formatting information.

C#

```
public string ToString (IFormatProvider? formatProvider);
```

Parameters

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of the value of the current DateTimeOffset object, as specified by `formatProvider`.

Exceptions

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by `formatProvider`.

Examples

The following example displays a DateTimeOffset object using CultureInfo objects that represent the invariant culture, as well as four other cultures.

C#

```
CultureInfo[] cultures = new CultureInfo[] {CultureInfo.InvariantCulture,
                                            new CultureInfo("en-us"),
                                            new CultureInfo("fr-fr"),
                                            new CultureInfo("de-DE"),
                                            new CultureInfo("es-ES")};

DateTimeOffset thisDate = new DateTimeOffset(2007, 5, 1, 9, 0, 0,
                                             TimeSpan.Zero);

foreach (CultureInfo culture in cultures)
{
   string cultureName;
   if (string.IsNullOrEmpty(culture.Name))
      cultureName = culture.NativeName;
   else
      cultureName = culture.Name;

   Console.WriteLine("In {0}, {1}",
                     cultureName, thisDate.ToString(culture));
}
// The example produces the following output:
//    In Invariant Language (Invariant Country), 05/01/2007 09:00:00
+00:00
//    In en-US, 5/1/2007 9:00:00 AM +00:00
//    In fr-FR, 01/05/2007 09:00:00 +00:00
//    In de-DE, 01.05.2007 09:00:00 +00:00
//    In es-ES, 01/05/2007 9:00:00 +00:00
```

Remarks

The return value of this method is identical to that of its equivalent overload of the DateTime.ToString method, except that it includes a space followed by the offset appended at the end of the string. In other words, it formats output using the short date pattern, the long time pattern, and the `zzz` custom format string, with each element separated from the previous element by a space.

The format of these three elements is defined by the `formatProvider` parameter. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture whose formatting conventions are applied to the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatProvider` is `null`, the DateTimeFormatInfo object associated with the current culture is used (see CurrentCulture).

Notes to Callers

The ToString(IFormatProvider) method returns the string representation of the date and time in the calendar used by the culture represented by the `formatProvider` parameter. Its calendar is defined by the Calendar property. If the value of the current DateTimeOffset instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the JapaneseCalendar class.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo jaJP = new CultureInfo("ja-JP");
      jaJP.DateTimeFormat.Calendar = new JapaneseCalendar();
      DateTimeOffset date1 = new DateTimeOffset(new DateTime(1867, 1, 1),
                                                TimeSpan.Zero);

      try {
         Console.WriteLine(date1.ToString(jaJP));
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("{0:d} is earlier than {1:d} or later than
{2:d}",
                           date1,

jaJP.DateTimeFormat.Calendar.MinSupportedDateTime,

jaJP.DateTimeFormat.Calendar.MaxSupportedDateTime);
```

```
        }
    }
}
// The example displays the following output:
//    1/1/1867 is earlier than 9/8/1868 or later than 12/31/9999
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A format string.

Returns

String

A string representation of the value of the current DateTimeOffset object, as specified by `format`.

Exceptions

FormatException

The length of `format` is one, and it is not one of the standard format specifier characters defined for DateTimeFormatInfo.

-or-

`format` does not contain a valid custom format pattern.

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by the current culture.

Examples

The following example displays a DateTimeOffset object to the console using each of the standard date and time format specifiers. The output is formatted by using the en-us culture.

C#

```
DateTimeOffset outputDate = new DateTimeOffset(2007, 10, 31, 21, 0, 0,
                                        new TimeSpan(-8, 0, 0));
string specifier;

// Output date using each standard date/time format specifier
specifier = "d";
// Displays   d: 10/31/2007
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "D";
// Displays   D: Wednesday, October 31, 2007
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "t";
// Displays   t: 9:00 PM
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "T";
// Displays   T: 9:00:00 PM
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "f";
// Displays   f: Wednesday, October 31, 2007 9:00 PM
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "F";
// Displays   F: Wednesday, October 31, 2007 9:00:00 PM
```

```
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "g";
// Displays   g: 10/31/2007 9:00 PM
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "G";
// Displays   G: 10/31/2007 9:00:00 PM
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "M";           // 'm' is identical
// Displays   M: October 31
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "R";           // 'r' is identical
// Displays   R: Thu, 01 Nov 2007 05:00:00 GMT
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "s";
// Displays   s: 2007-10-31T21:00:00
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

specifier = "u";
// Displays   u: 2007-11-01 05:00:00Z
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));

// Specifier is not supported
specifier = "U";
try
{
   Console.WriteLine("{0}: {1}", specifier,
outputDate.ToString(specifier));
}
catch (FormatException)
{
   Console.WriteLine("{0}: Not supported.", specifier);
}

specifier = "Y";         // 'y' is identical
// Displays   Y: October, 2007
Console.WriteLine("{0}: {1}", specifier, outputDate.ToString(specifier));
```

Remarks

The `format` parameter should contain either a single format specifier character (see Standard Date and Time Format Strings) or a custom format pattern (see Custom Date and Time Format Strings) that defines the format of the returned string. If `format` is a null or empty string ("""), the DateTimeOffset value is output using the default format.

The following table shows the exact operation of certain format specifiers when used with DateTimeOffset, which differs from their behavior when used with DateTime.

Expand table

Existing format specifier	New behavior
"K"	Designed to round-trip a date and time. With DateTimeOffset, maps to "zzz" (the offset is always displayed with hours and minutes). Note that "K" is a custom format specifier; it cannot appear as the single character in `format`.
"U"	Not supported.
"r"	Converts the DateTimeOffset object to Coordinated Universal Time (UTC) and outputs it using the custom format string `ddd, dd MMM yyyy HH:mm:ss GMT`.
"u"	Converts the DateTimeOffset object to UTC and outputs it using the format `yyyy-MM-dd HH:mm:ssZ`.

The remaining standard date and time format specifiers behave the same with the ToString(String) method as they do with the ToString method.

This method uses formatting information derived from the current culture. For more information, see CurrentCulture.

Notes to Callers

The ToString(String) method returns the string representation of the date and time in the calendar used by the current culture. If the value of the current DateTimeOffset instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the HebrewCalendar class when the current culture is Hebrew (Israel).

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
   public static void Main()
   {
```

```
      DateTimeOffset date1 = new DateTimeOffset(new DateTime(1550, 7,
21),
                                                TimeSpan.Zero);
      CultureInfo dft;
      CultureInfo heIL = new CultureInfo("he-IL");
      heIL.DateTimeFormat.Calendar = new HebrewCalendar();

      // Change current culture to he-IL.
      dft = Thread.CurrentThread.CurrentCulture;
      Thread.CurrentThread.CurrentCulture = heIL;

      // Display the date using the current culture's calendar.
      try {
         Console.WriteLine(date1.ToString("G"));
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("{0} is earlier than {1} or later than {2}",
                           date1.ToString("d",
CultureInfo.InvariantCulture),

heIL.DateTimeFormat.Calendar.MinSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture),

heIL.DateTimeFormat.Calendar.MaxSupportedDateTime.ToString("d",
CultureInfo.InvariantCulture));
      }

      // Restore the default culture.
      Thread.CurrentThread.CurrentCulture = dft;
   }
}
// The example displays the following output:
//    07/21/1550 is earlier than 01/01/1583 or later than 09/29/2239
```

See also

- Standard Date and Time Format Strings
- Custom Date and Time Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the value of the current DateTimeOffset object to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? formatProvider);
```

Parameters

`format` String

A format string.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

A string representation of the value of the current DateTimeOffset object, as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The length of `format` is one, and it is not one of the standard format specifier characters defined for DateTimeFormatInfo.

-or-

`format` does not contain a valid custom format pattern.

ArgumentOutOfRangeException

The date and time is outside the range of dates supported by the calendar used by `formatProvider`.

Examples

The following example uses the ToString(String, IFormatProvider) method to display a DateTimeOffset object using a custom format string for several different cultures.

C#

```
DateTimeOffset outputDate = new DateTimeOffset(2007, 11, 1, 9, 0, 0,
                                       new TimeSpan(-7, 0, 0));
string format = "dddd, MMM dd yyyy HH:mm:ss zzz";

// Output date and time using custom format specification
Console.WriteLine(outputDate.ToString(format, null as
DateTimeFormatInfo));
Console.WriteLine(outputDate.ToString(format,
CultureInfo.InvariantCulture));
Console.WriteLine(outputDate.ToString(format,
                                        new CultureInfo("fr-FR")));
Console.WriteLine(outputDate.ToString(format,
                                        new CultureInfo("es-ES")));
// The example displays the following output to the console:
//    Thursday, Nov 01 2007 09:00:00 -07:00
//    Thursday, Nov 01 2007 09:00:00 -07:00
//    jeudi, nov. 01 2007 09:00:00 -07:00
//    jueves, nov 01 2007 09:00:00 -07:00
```

Remarks

The `format` parameter should contain either a single format specifier character (see Standard Date and Time Format Strings) or a custom format pattern (see Custom Date and Time Format Strings). If `format` is a null or empty string ("), the DateTimeOffset object is output using the default format.

The following table shows the exact operation of certain format specifiers when used with DateTimeOffset, which differs from their behavior when used with DateTime.

Expand table

Existing format specifier	New behavior
"K"	Designed to round-trip a date and time. With DateTimeOffset, maps to "zzz" (the offset is always displayed with hours and minutes). Note that "K" is a custom format specifier; it cannot appear as the single character in `format`.
"U"	Not supported.
"r"	Converts the DateTimeOffset object to Coordinated Universal Time (UTC) and outputs it using the custom format string `ddd, dd MMM yyyy HH:mm:ss GMT`.
"u"	Converts the DateTimeOffset value to UTC and outputs it using the format `yyyy-MM-dd HH:mm:ssZ`.

The remaining standard date and time format specifiers behave the same with the ToString(String) method as they do with the ToString method.

The pattern that corresponds to standard format specifiers, as well as the symbols and names of date and time components, is defined by the `formatProvider` parameter. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture whose formatting is used in `input`. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting used in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatProvider` is `null`, the DateTimeFormatInfo object associated with the current culture is used (see CurrentCulture).

Notes to Callers

The ToString(String, IFormatProvider) method returns the string representation of the date and time in the calendar used by the `formatProvider` parameter. Its calendar is defined by the Calendar property. If the value of the current DateTimeOffset instance is earlier than MinSupportedDateTime or later than MaxSupportedDateTime, the method throws an ArgumentOutOfRangeException. The following example provides an illustration. It attempts to format a date that is outside the range of the UmAlQuraCalendar class.

```
C#
```

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      CultureInfo arSA = new CultureInfo("ar-SA");
      arSA.DateTimeFormat.Calendar = new UmAlQuraCalendar();
      DateTimeOffset date1 = new DateTimeOffset(new DateTime(1890, 9,
10),
                                                TimeSpan.Zero);

      try {
         Console.WriteLine(date1.ToString("d", arSA));
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("{0:d} is earlier than {1:d} or later than
{2:d}",
                           date1,

arSA.DateTimeFormat.Calendar.MinSupportedDateTime,

arSA.DateTimeFormat.Calendar.MaxSupportedDateTime);
      }
   }
}
// The example displays the following output:
//    9/10/1890 is earlier than 4/30/1900 or later than 5/13/2029
```

See also

- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToUniversalTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current DateTimeOffset object to a DateTimeOffset value that represents the Coordinated Universal Time (UTC).

C#

```
public DateTimeOffset ToUniversalTime ();
```

Returns

DateTimeOffset

An object that represents the date and time of the current DateTimeOffset object converted to Coordinated Universal Time (UTC).

Examples

The following example calls the ToUniversalTime method to convert a local time and several other times to Coordinated Universal Time (UTC).

C#

```
DateTimeOffset localTime, otherTime, universalTime;

// Define local time in local time zone
localTime = new DateTimeOffset(new DateTime(2007, 6, 15, 12, 0, 0));
Console.WriteLine("Local time: {0}", localTime);
Console.WriteLine();

// Convert local time to offset 0 and assign to otherTime
otherTime = localTime.ToOffset(TimeSpan.Zero);
Console.WriteLine("Other time: {0}", otherTime);
Console.WriteLine("{0} = {1}: {2}",
                  localTime, otherTime,
```

```
                    localTime.Equals(otherTime));
Console.WriteLine("{0} exactly equals {1}: {2}",
                    localTime, otherTime,
                    localTime.EqualsExact(otherTime));
Console.WriteLine();

// Convert other time to UTC
universalTime = localTime.ToUniversalTime();
Console.WriteLine("Universal time: {0}", universalTime);
Console.WriteLine("{0} = {1}: {2}",
                    otherTime, universalTime,
                    universalTime.Equals(otherTime));
Console.WriteLine("{0} exactly equals {1}: {2}",
                    otherTime, universalTime,
                    universalTime.EqualsExact(otherTime));
Console.WriteLine();
// The example produces the following output to the console:
//    Local time: 6/15/2007 12:00:00 PM -07:00
//
//    Other time: 6/15/2007 7:00:00 PM +00:00
//    6/15/2007 12:00:00 PM -07:00 = 6/15/2007 7:00:00 PM +00:00: True
//    6/15/2007 12:00:00 PM -07:00 exactly equals 6/15/2007 7:00:00 PM
+00:00: False
//
//    Universal time: 6/15/2007 7:00:00 PM +00:00
//    6/15/2007 7:00:00 PM +00:00 = 6/15/2007 7:00:00 PM +00:00: True
//    6/15/2007 7:00:00 PM +00:00 exactly equals 6/15/2007 7:00:00 PM
+00:00: True
```

Remarks

This method returns a DateTimeOffset object whose Offset property is set to zero.

Both the value of the current DateTimeOffset object and the value of the DateTimeOffset object returned by the method call represent the same point in time. That is, if both are passed to the DateTimeOffset.Equals(DateTimeOffset, DateTimeOffset) method, the method will return `true`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- ToLocalTime()
- ToOffset(TimeSpan)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToUnixTimeMilliseconds Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of milliseconds that have elapsed since 1970-01-01T00:00:00.000Z.

C#

```
public long ToUnixTimeMilliseconds ();
```

Returns

Int64

The number of milliseconds that have elapsed since 1970-01-01T00:00:00.000Z.

Remarks

Unix time represents the number of seconds that have elapsed since 1970-01-01T00:00:00Z (January 1, 1970, at 12:00 AM UTC). It does not take leap seconds into account. This method returns the number of milliseconds in Unix time.

This method first converts the current instance to UTC before returning the number of milliseconds in its Unix time. For date and time values before 1970-01-01T00:00:00Z, this method returns a negative value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- ToUnixTimeSeconds()
- FromUnixTimeMilliseconds(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.ToUnixTimeSeconds Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of seconds that have elapsed since 1970-01-01T00:00:00Z.

C#

```
public long ToUnixTimeSeconds ();
```

Returns

Int64

The number of seconds that have elapsed since 1970-01-01T00:00:00Z.

Examples

The following example calls the ToUnixTimeSeconds method to return the Unix time of values that are equal to, shortly before, and shortly after 1970-01-01T00:00:00Z.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      DateTimeOffset dto = new DateTimeOffset(1970, 1, 1, 0, 0, 0,
TimeSpan.Zero);
      Console.WriteLine("{0} --> Unix Seconds: {1}", dto,
dto.ToUnixTimeSeconds());

      dto = new DateTimeOffset(1969, 12, 31, 23, 59, 0, TimeSpan.Zero);
      Console.WriteLine("{0} --> Unix Seconds: {1}", dto,
dto.ToUnixTimeSeconds());
```

```
        dto = new DateTimeOffset(1970, 1, 1, 0, 1, 0, TimeSpan.Zero);
        Console.WriteLine("{0} --> Unix Seconds: {1}", dto,
dto.ToUnixTimeSeconds());
    }
}
// The example displays the following output:
//    1/1/1970 12:00:00 AM +00:00 --> Unix Seconds: 0
//    12/31/1969 11:59:00 PM +00:00 --> Unix Seconds: -60
//    1/1/1970 12:01:00 AM +00:00 --> Unix Seconds: 60
```

Remarks

Unix time represents the number of seconds that have elapsed since 1970-01-01T00:00:00Z (January 1, 1970, at 12:00 AM UTC). It does not take leap seconds into account.

This method first converts the current instance to UTC before returning its Unix time. For date and time values before 1970-01-01T00:00:00Z, this method returns a negative value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FromUnixTimeSeconds(Int64)
- ToUnixTimeMilliseconds()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

DateTimeOffset.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current datetime offset instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? formatProvider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`formatProvider` IFormatProvider

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current datetime offset instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? formatProvider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the charactes that represent a standard or custom format string that defines the acceptable format for `destination`.

`formatProvider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

DateTimeOffset.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified string representation of a date and time to its DateTimeOffset equivalent.

Overloads

Expand table

TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset)	Tries to convert a specified string representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)	Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateTimeOffset)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeOffset)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, DateTimeOffset)	Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, DateTimeOffset)	Tries to converts a specified string representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.

TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset)

Tries to convert a specified string representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (string? input, IFormatProvider?
formatProvider, System.Globalization.DateTimeStyles styles, out
DateTimeOffset result);
```

Parameters

`input` String

A string that contains a date and time to convert.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset value equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` parameter is `null` or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Exceptions

ArgumentException

`styles` includes an undefined DateTimeStyles value.

-or-

NoCurrentDateDefault is not supported.

-or-

`styles` includes mutually exclusive DateTimeStyles values.

Examples

The following example calls the TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset) method with a variety of DateTimeStyles values to parse some strings with various date and time formats.

C#

```
string dateString;
DateTimeOffset parsedDate;

dateString = "05/01/2008 6:00:00";
// Assume time is local
if (DateTimeOffset.TryParse(dateString, null as IFormatProvider,
                            DateTimeStyles.AssumeLocal,
                            out parsedDate))
   Console.WriteLine("'{0}' was converted to {1}.",
                     dateString, parsedDate.ToString());
else
   Console.WriteLine("Unable to parse '{0}'.", dateString);

// Assume time is UTC
if (DateTimeOffset.TryParse(dateString, null as IFormatProvider,
                            DateTimeStyles.AssumeUniversal,
                            out parsedDate))
   Console.WriteLine("'{0}' was converted to {1}.",
                     dateString, parsedDate.ToString());
else
   Console.WriteLine("Unable to parse '{0}'.", dateString);

// Parse and convert to UTC
dateString = "05/01/2008 6:00:00AM +5:00";
if (DateTimeOffset.TryParse(dateString, null as IFormatProvider,
                           DateTimeStyles.AdjustToUniversal,
                           out parsedDate))
   Console.WriteLine("'{0}' was converted to {1}.",
                     dateString, parsedDate.ToString());
else
   Console.WriteLine("Unable to parse '{0}'.", dateString);
// The example displays the following output to the console:
//    '05/01/2008 6:00:00' was converted to 5/1/2008 6:00:00 AM -07:00.
```

```
//    '05/01/2008 6:00:00' was converted to 5/1/2008 6:00:00 AM +00:00.
//    '05/01/2008 6:00:00AM +5:00' was converted to 5/1/2008 1:00:00 AM
+00:00.
```

Remarks

This overload of the TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset) method is like the DateTimeOffset.Parse(String, IFormatProvider, DateTimeStyles) method, except that it does not throw an exception if the conversion fails. The method parses a string with three elements that can appear in any order and are delimited by white space. These three elements are shown in the following table.

Expand table

Element	Example
<Date>	"2/10/2007"
<Time>	"1:02:03 PM"
<Offset>	"-7:30"

Although each of these elements is optional, <Offset> cannot appear by itself. It must be provided together with either <Date> or <Time>. If <Date> is missing, its default value is the current day. If <Date> is present but its year component consists of only two digits, it is converted to a year in the `provider` parameter's current calendar based on the value of the Calendar.TwoDigitYearMax property. If <Time> is missing, its default value is 12:00:00 AM. If <Offset> is missing, its default value is the offset of the local time zone, or Zero if either the DateTimeStyles.AdjustToUniversal or DateTimeStyles.AssumeUniversal value is specified in `styles`. If <Offset> is present, it can represent either a negative or a positive offset from Coordinated Universal Time (UTC). In either case, <Offset> must include a sign symbol or the method returns `false`.

The `input` string is parsed by using the culture-specific formatting information in a DateTimeFormatInfo object supplied by the `formatProvider` parameter. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture whose formatting is used in `input`. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the format that is used in `input`.

- A DateTimeFormatInfo object that defines the format of date and time data.

In addition, each element can be delimited by leading or trailing white space, and the <Date> and <Time> components can include inner white space (such as 6: 00:00). Only the <Offset> component cannot include inner white space.

If `provider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The positive or negative sign used in <Offset> must be either + or -. It is not defined by the PositiveSign or NegativeSign properties of the NumberFormatInfo object returned by the `formatprovider` parameter's NumberFormat property.

The following members of the DateTimeStyles enumeration are supported:

Expand table

DateTimeStyles Member	Comments
AdjustToUniversal	Parses the string represented by `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the returned object's ToUniversalTime() method.
AllowInnerWhite	Although valid, this value is ignored. Inner white space is allowed in the <Date> and <Time> components.
AllowLeadingWhite	Although valid, this value is ignored. Leading white space is allowed in front of each component in the parsed string.
AllowTrailingWhite	Although valid, this value is ignored. Trailing white space is allowed in front of each component in the parsed string.
AllowWhiteSpaces	This is the default behavior. It cannot be overridden by supplying a more restrictive DateTimeStyles enumeration value, such as DateTimeStyles.None.
AssumeLocal	Indicates that, if the `input` parameter lacks an <Offset> element, the offset of the local time zone should be provided. This is the default behavior of the TryParse(String, IFormatProvider, DateTimeStyles, DateTimeOffset) method.
AssumeUniversal	Indicates that, if the `input` parameter lacks an <Offset> element, the UTC offset (00:00) should be provided.
None	Although valid, this value is ignored and has no effect.
RoundtripKind	Because the DateTimeOffset structure does not include a `Kind` property, this value has no effect.

Only the DateTimeStyles.NoCurrentDateDefault value is not supported. An ArgumentException is thrown if this value is included in the `styles` parameter.

See also

- Parse

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)

Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> input, IFormatProvider?
formatProvider, System.Globalization.DateTimeStyles styles, out
DateTimeOffset result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters representing the date and time to convert.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset value equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` parameter is `null` or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, IFormatProvider, DateTimeOffset)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider, out
DateTimeOffset result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateTimeOffset

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeOffset)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out DateTimeOffset result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` DateTimeOffset

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, DateTimeOffset)

Tries to convert a specified span representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> input, out DateTimeOffset
result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters representing the date and time to convert.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` parameter is `null` or does not contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, DateTimeOffset)

Tries to converts a specified string representation of a date and time to its DateTimeOffset equivalent, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? input, out DateTimeOffset result);
```

Parameters

`input` String

A string that contains a date and time to convert.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` parameter is `null` or does not

contain a valid string representation of a date and time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Examples

The following example calls the TryParse(String, DateTimeOffset) method to parse several strings with various date and time formats.

C#

```
DateTimeOffset parsedDate;
string dateString;

// String with date only
dateString = "05/01/2008";
if (DateTimeOffset.TryParse(dateString, out parsedDate))
   Console.WriteLine("{0} was converted to {1}.",
                     dateString, parsedDate);

// String with time only
dateString = "11:36 PM";
if (DateTimeOffset.TryParse(dateString, out parsedDate))
   Console.WriteLine("{0} was converted to {1}.",
                     dateString, parsedDate);

// String with date and offset
dateString = "05/01/2008 +7:00";
if (DateTimeOffset.TryParse(dateString, out parsedDate))
   Console.WriteLine("{0} was converted to {1}.",
                     dateString, parsedDate);

// String with day abbreviation
dateString = "Thu May 01, 2008";
if (DateTimeOffset.TryParse(dateString, out parsedDate))
   Console.WriteLine("{0} was converted to {1}.",
                     dateString, parsedDate);

// String with date, time with AM/PM designator, and offset
dateString = "5/1/2008 10:00 AM -07:00";
if (DateTimeOffset.TryParse(dateString, out parsedDate))
   Console.WriteLine("{0} was converted to {1}.",
                     dateString, parsedDate);
// if (run on 3/29/07, the example displays the following output
// to the console:
//    05/01/2008 was converted to 5/1/2008 12:00:00 AM -07:00.
```

```
//    11:36 PM was converted to 3/29/2007 11:36:00 PM -07:00.
//    05/01/2008 +7:00 was converted to 5/1/2008 12:00:00 AM +07:00.
//    Thu May 01, 2008 was converted to 5/1/2008 12:00:00 AM -07:00.
//    5/1/2008 10:00 AM -07:00 was converted to 5/1/2008 10:00:00 AM
-07:00.
```

Remarks

This overload of the TryParse(String, DateTimeOffset) method is like the DateTimeOffset.Parse(String) method, except that it does not throw an exception if the conversion fails. It parses a string with three elements that can appear in any order and are delimited by white space. These three elements are shown in the following table.

Expand table

Element	Example
<Date>	"2/10/2007"
<Time>	"1:02:03 PM"
<Offset>	"-7:30"

Although each of these elements is optional, <Offset> cannot appear by itself. It must be provided together with either <Date> or <Time>. If <Date> is missing, its default value is the current day. If <Date> is present but its year component consists of only two digits, it is converted to a year in the current culture's current calendar based on the value of the Calendar.TwoDigitYearMax property. If <Time> is missing, its default value is 12:00:00 AM. If <Offset> is missing, its default value is the offset of the local time zone. If <Offset> is present, it can represent either a negative or a positive offset from Coordinated Universal Time (UTC). In either case, <Offset> must include a sign symbol or the method returns `false`.

The `input` string is parsed by using the formatting information in a DateTimeFormatInfo object initialized for the current culture. To parse a string that contains designated formatting that does not necessarily correspond to that of the current culture, use the TryParseExact method and provide a format specifier.

See also

- Parse
- Sample: .NET Core WinForms Formatting Utility (C#)

- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a date and time to its DateTimeOffset equivalent. The format of the string representation must match a specified format exactly.

Overloads

Expand table

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTimeOffset)	Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (string? input, string?[]? formats,
IFormatProvider? formatProvider, System.Globalization.DateTimeStyles
styles, out DateTimeOffset result);
```

Parameters

`input` String

A string that contains a date and time to convert.

`formats` String[]

An array that defines the expected formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of input. A typical value to specify is `None`.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` does not contain a valid string representation of a date and time, or does not contain the date and time in the expected format defined by `format`, or if `formats` is `null`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Exceptions

ArgumentException

`styles` includes an undefined DateTimeStyles value.

-or-

NoCurrentDateDefault is not supported.

-or-

`styles` includes mutually exclusive DateTimeStyles values.

Examples

The following example defines multiple input formats for the string representation of a date and time and offset value, and then passes the string that is entered by the user to the TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTimeOffset) method.

C#

```
TextReader conIn = Console.In;
TextWriter conOut = Console.Out;
int tries = 0;
string input = String.Empty;

string[] formats = new string[] {"M/dd/yyyy HH:m zzz", "MM/dd/yyyy HH:m
zzz",
                                 "M/d/yyyy HH:m zzz", "MM/d/yyyy HH:m
zzz",
                                 "M/dd/yy HH:m zzz", "MM/dd/yy HH:m zzz",
                                 "M/d/yy HH:m zzz", "MM/d/yy HH:m zzz",
                                 "M/dd/yyyy H:m zzz", "MM/dd/yyyy H:m
zzz",
                                 "M/d/yyyy H:m zzz", "MM/d/yyyy H:m zzz",
                                 "M/dd/yy H:m zzz", "MM/dd/yy H:m zzz",
                                 "M/d/yy H:m zzz", "MM/d/yy H:m zzz",
                                 "M/dd/yyyy HH:mm zzz", "MM/dd/yyyy HH:mm
zzz",
                                 "M/d/yyyy HH:mm zzz", "MM/d/yyyy HH:mm
zzz",
                                 "M/dd/yy HH:mm zzz", "MM/dd/yy HH:mm
zzz",
                                 "M/d/yy HH:mm zzz", "MM/d/yy HH:mm zzz",
                                 "M/dd/yyyy H:mm zzz", "MM/dd/yyyy H:mm
zzz",
```

```
                                    "M/d/yyyy H:mm zzz", "MM/d/yyyy H:mm
zzz",
                                    "M/dd/yy H:mm zzz", "MM/dd/yy H:mm zzz",
                                    "M/d/yy H:mm zzz", "MM/d/yy H:mm zzz"};
IFormatProvider provider = CultureInfo.InvariantCulture.DateTimeFormat;
DateTimeOffset result;

do {
   conOut.WriteLine("Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-
HH:MM),");
   conOut.Write("Then press Enter: ");
   input = conIn.ReadLine();
   conOut.WriteLine();
   if (DateTimeOffset.TryParseExact(input, formats, provider,
                                    DateTimeStyles.AllowWhiteSpaces,
                                    out result))
   {
      break;
   }
   else
   {
      Console.WriteLine("Unable to parse {0}.", input);
      tries++;
   }
} while (tries < 3);
if (tries >= 3)
   Console.WriteLine("Exiting application without parsing {0}", input);
else
   Console.WriteLine("{0} was converted to {1}", input,
result.ToString());
// Some successful sample interactions with the user might appear as
follows:
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/08/2007 6:54 -6:00
//
//    12/08/2007 6:54 -6:00 was converted to 12/8/2007 6:54:00 AM -06:00
//
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/8/2007 06:54 -06:00
//
//    12/8/2007 06:54 -06:00 was converted to 12/8/2007 6:54:00 AM -06:00
//
//    Enter a date, time, and offset (MM/DD/YYYY HH:MM +/-HH:MM),
//    Then press Enter: 12/5/07 6:54 -6:00
//
//    12/5/07 6:54 -6:00 was converted to 12/5/2007 6:54:00 AM -06:00
```

Remarks

The TryParseExact(String, String[], IFormatProvider, DateTimeStyles, DateTimeOffset) method parses the string representation of a date that matches any one of the patterns assigned to the `formats` array. If the `input` string does not match any one

of these patterns with any variations defined by the `styles` parameter, the parsing operation fails and the method returns `false`. Aside from comparing `input` to multiple strings that contain format specifiers, this overload behaves identically to the DateTimeOffset.ParseExact(String, String[], IFormatProvider, DateTimeStyles) method.

The `formats` parameter is a string array whose elements contain either a single standard format specifier or one or more custom format specifiers that define the possible pattern of `input`. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If the matched element in `formats` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the parse operation fails and the method returns `false`.

> **ⓘ Important**
>
> Using the `formats` parameter of this overload to specify multiple formats can help reduce the frustration many users experience when they enter dates and times. In particular, the ability to define multiple input patterns enables an application to handle date and time representations that can either include or lack leading zeros in months, days, hours, minutes, and seconds. The example provides an illustration of this.

If the matched element in `formats` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If the matched element in `formats` requires that input contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If the matched element in `formats` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise pattern of `input` if the matching element of `formats` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and formatting in `input`.

- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is permitted in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default value.
AssumeUniversal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by the elements in `formats`. Extra white space can appear between date and time components and within individual components (except for the offset) and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by the elements in `formats`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by the elements in `formats`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by the elements in `formats`. All extra white-space characters not

`DateTimeStyles` member	Behavior
	specified in the matched element in `formats` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in a particular element in `formats` for a successful match to occur. This is the default behavior.
RoundtripKind	Has no effect because the DateTimeOffset structure does not include a `Kind` property.

Notes to Callers

In the .NET Framework 4, the TryParseExact returns `false` if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, DateTimeOffset)

Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the date and time representation must match the specified format exactly.

```C#
public static bool TryParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, IFormatProvider? formatProvider,
System.Globalization.DateTimeStyles styles, out DateTimeOffset result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a date and time to convert.

`format` ReadOnlySpan<Char>

A format specifier that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`. A typical value to specify is None.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue if the conversion failed. The conversion fails if the

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Exceptions

ArgumentException

`styles` includes an undefined DateTimeStyles value. -or- NoCurrentDateDefault is not supported. -or- `styles` includes mutually exclusive DateTimeStyles values.

Remarks

This overload is like the DateTimeOffset.ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles) method, except that this method does not throw an exception if the conversion fails. It parses the representation of a date and time that must exactly match the pattern specified by the `format` parameter. If `input` does not match this pattern, with some possible variations in white space defined by the `styles` parameter, the parsing operation fails and the method returns `false`.

The `format` parameter is a character span that contains either a single standard format specifier or one or more custom format specifiers that define the required pattern of `input`. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If `format` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the parsing operation fails and the method returns `false`.

If `format` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If `format` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If `format` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise pattern of `input` if `format` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and standard formats allowed in `input`.

- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is allowed in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a date and time representation, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default behavior.
AssumeUniversal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by format. Extra white space can appear between date and time components and within individual components, other than the offset, and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by `format`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by `format`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by `format`. All extra white-space characters not specified in `format` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in `format`. This is the default behavior.
RoundtripKind	Has no effect, because the DateTimeOffset structure does not include a `Kind` property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, DateTimeOffset)

Converts the representation of a date and time in a character span to its DateTimeOffset equivalent using the specified formats, culture-specific format information, and style. The format of the date and time representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input, string?[]?
formats, IFormatProvider? formatProvider,
System.Globalization.DateTimeStyles styles, out DateTimeOffset result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a date and time to convert.

`formats` String[]

A array of standard or custom format strings that define the acceptable formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `input`. A typical value to specify is None.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue if the conversion failed. The conversion fails if the

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Exceptions

ArgumentException

`styles` includes an undefined DateTimeStyles value. -or- NoCurrentDateDefault is not supported. -or- `styles` includes mutually exclusive DateTimeStyles values.

Remarks

This method parses the string representation of a date that matches any one of the patterns assigned to the `formats` array. If `input` does not match any one of these patterns with any variations defined by the `styles` parameter, the parsing operation fails and the method returns `false`. Aside from comparing `input` to multiple strings that contain format specifiers, this overload behaves identically to the DateTimeOffset.ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles) method.

The `formats` parameter is a string array whose elements contain either a single standard format specifier or one or more custom format specifiers that define the possible pattern of `input`. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If the matched element in `formats` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the parse operation fails and the method returns `false`.

> **ⓘ Important**
>
> Using the `formats` parameter of this overload to specify multiple formats can help reduce the frustration many users experience when they enter dates and times. In particular, the ability to define multiple input patterns enables an application to handle date and time representations that can either include or

lack leading zeros in months, days, hours, minutes, and seconds. The example provides an illustration of this.

If the matched element in `formats` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If the matched element in `formats` requires that input contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If the matched element in `formats` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes DateTimeStyles.AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes DateTimeStyles.AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols used in `input` are defined by the `formatProvider` parameter. The same is true for the precise pattern of `input` if the matching element of `formats` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and formatting in `input`.
- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is permitted in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default value.
AssumeUniversal	If the matched element in `formats` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by the elements in `formats`. Extra white space can appear between date and time components and within individual components (except for the offset) and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by the elements in `formats`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by the elements in `formats`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by the elements in `formats`. All extra white-space characters not specified in the matched element in `formats` are ignored when parsing the string.
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in a particular element in `formats` for a successful match to occur. This is the default behavior.
RoundtripKind	Has no effect because the DateTimeOffset structure does not include a `Kind` property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTimeOffset)

Converts the specified string representation of a date and time to its DateTimeOffset equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly.

C#

```
public static bool TryParseExact (string? input, string? format,
IFormatProvider? formatProvider, System.Globalization.DateTimeStyles
styles, out DateTimeOffset result);
```

Parameters

`input` String

A string that contains a date and time to convert.

`format` String

A format specifier that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information about `input`.

`styles` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of input. A typical value to specify is `None`.

`result` DateTimeOffset

When the method returns, contains the DateTimeOffset equivalent to the date and time of `input`, if the conversion succeeded, or DateTimeOffset.MinValue, if the conversion failed. The conversion fails if the `input` parameter is `null`, or does not contain a valid string representation of a date and time in the expected format defined by `format` and `provider`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `input` parameter is successfully converted; otherwise, `false`.

Exceptions

ArgumentException

`styles` includes an undefined DateTimeStyles value.

-or-

NoCurrentDateDefault is not supported.

-or-

`styles` includes mutually exclusive DateTimeStyles values.

Examples

The following example uses the TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTimeOffset) method with standard and custom format specifiers, the invariant culture, and various DateTimeStyles values to parse several date and time strings.

C#

```
string dateString, format;
DateTimeOffset result;
IFormatProvider provider = CultureInfo.InvariantCulture;

// Parse date-only value with invariant culture and assume time is UTC.
dateString = "06/15/2008";
format = "d";
if (DateTimeOffset.TryParseExact(dateString, format, provider,
                                 DateTimeStyles.AssumeUniversal,
                                 out result))
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
else
   Console.WriteLine("'{0}' is not in the correct format.", dateString);

// Parse date-only value with leading white space.
// Should return False because only trailing white space is
// specified in method call.
dateString = " 06/15/2008";
if (DateTimeOffset.TryParseExact(dateString, format, provider,
                                 DateTimeStyles.AllowTrailingWhite,
                                 out result))
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
else
   Console.WriteLine("'{0}' is not in the correct format.", dateString);

// Parse date and time value, and allow all white space.
```

```
dateString = " 06/15/   2008  15:15    -05:00";
format = "MM/dd/yyyy H:mm zzz";
if (DateTimeOffset.TryParseExact(dateString, format, provider,
                                 DateTimeStyles.AllowWhiteSpaces,
                                 out result))
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
else
   Console.WriteLine("'{0}' is not in the correct format.", dateString);

// Parse date and time and convert to UTC.
dateString = "  06/15/2008 15:15:30 -05:00";
format = "MM/dd/yyyy H:mm:ss zzz";
if (DateTimeOffset.TryParseExact(dateString, format, provider,
                                 DateTimeStyles.AllowWhiteSpaces |
                                 DateTimeStyles.AdjustToUniversal,
                                 out result))
   Console.WriteLine("'{0}' converts to {1}.", dateString,
result.ToString());
else
   Console.WriteLine("'{0}' is not in the correct format.", dateString);
// The example displays the following output:
//    '06/15/2008' converts to 6/15/2008 12:00:00 AM +00:00.
//    ' 06/15/2008' is not in the correct format.
//    ' 06/15/   2008  15:15    -05:00' converts to 6/15/2008 3:15:00 PM
-05:00.
//    '  06/15/2008 15:15:30 -05:00' converts to 6/15/2008 8:15:30 PM
+00:00.
```

The following example uses a variety of DateTimeStyles values to parse an array of strings that are expected to conform to ISO 8601. As the output from the example shows, strings that are in the proper format fail to parse if:

- they contain white space, and an appropriate DateTimeStyles flag (such as DateTimeStyles.AllowWhiteSpaces has not been supplied in the method call.
- they contain date and time elements that are out of range.

Strings that do not specify a UTC offset are assumed to have the offset of the local time zone (in this case, -07:00, which reflects the offset of the Pacific Daylight Time zone) unless the DateTimeStyles.AssumeUniversal flag is supplied in the method call. In that case, they are assumed to be Universal Coordinated Time.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
```

```
    {
        string[] dateStrings = { "2018-08-18T12:45:16.0000000Z",
                                 "2018/08/18T12:45:16.0000000Z",
                                 "2018-18-08T12:45:16.0000000Z",
                                 "2018-08-18T12:45:16.0000000",
                                 " 2018-08-18T12:45:16.0000000Z ",
                                 "2018-08-18T12:45:16.0000000+02:00",
                                 "2018-08-18T12:45:16.0000000-07:00" };

        ParseWithISO8601(dateStrings, DateTimeStyles.None);
        Console.WriteLine("\n-----\n");
        ParseWithISO8601(dateStrings, DateTimeStyles.AllowWhiteSpaces);
        Console.WriteLine("\n-----\n");
        ParseWithISO8601(dateStrings, DateTimeStyles.AdjustToUniversal);
        Console.WriteLine("\n-----\n");
        ParseWithISO8601(dateStrings, DateTimeStyles.AssumeLocal);
        Console.WriteLine("\n-----\n");
        ParseWithISO8601(dateStrings, DateTimeStyles.AssumeUniversal);   }

    private static void ParseWithISO8601(string[] dateStrings,
DateTimeStyles styles)
    {
        Console.WriteLine($"Parsing with {styles}:");
        DateTimeOffset date;
        foreach (var dateString in dateStrings)
        {
            if (DateTimeOffset.TryParseExact(dateString, "O", null, styles,
out date))
            {
                Console.WriteLine($"   {dateString,-35} --> {date:yyyy-MM-dd
HH:mm:ss.FF zzz}");
            }
            else
            {
                Console.WriteLine($"   Unable to convert '{dateString}'");
            }
        }
    }
}
// The example displays the following output:
//      Parsing with None:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         Unable to convert '2018/08/18T12:45:16.0000000Z'
//         Unable to convert '2018-18-08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//         Unable to convert ' 2018-08-18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
//
//      -----
//
```

```
//      Parsing with AllowWhiteSpaces:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         Unable to convert '2018/08/18T12:45:16.0000000Z'
//         Unable to convert '2018-18-08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//          2018-08-18T12:45:16.0000000Z       --> 2018-08-18 12:45:16
+00:00
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
//
//      -----
//
//      Parsing with AdjustToUniversal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         Unable to convert '2018/08/18T12:45:16.0000000Z'
//         Unable to convert '2018-18-08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 19:45:16
+00:00
//         Unable to convert ' 2018-08-18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 10:45:16
+00:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 19:45:16
+00:00
//
//      -----
//
//      Parsing with AssumeLocal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         Unable to convert '2018/08/18T12:45:16.0000000Z'
//         Unable to convert '2018-18-08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
-07:00
//         Unable to convert ' 2018-08-18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
+02:00
//         2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
//
//      -----
//
//      Parsing with AssumeUniversal:
//         2018-08-18T12:45:16.0000000Z        --> 2018-08-18 12:45:16
+00:00
//         Unable to convert '2018/08/18T12:45:16.0000000Z'
//         Unable to convert '2018-18-08T12:45:16.0000000Z'
//         2018-08-18T12:45:16.0000000         --> 2018-08-18 12:45:16
+00:00
//         Unable to convert ' 2018-08-18T12:45:16.0000000Z '
//         2018-08-18T12:45:16.0000000+02:00   --> 2018-08-18 12:45:16
```

```
+02:00
//          2018-08-18T12:45:16.0000000-07:00   --> 2018-08-18 12:45:16
-07:00
```

Remarks

This overload of the TryParseExact(String, String, IFormatProvider, DateTimeStyles, DateTimeOffset) method is like the DateTimeOffset.ParseExact(String, String, IFormatProvider, DateTimeStyles) method, except that this method does not throw an exception if the conversion fails. It parses the string representation of a date and time that must exactly match the pattern specified by the `format` parameter. If the `input` string does not match this pattern, with some possible variations in white space defined by the `styles` parameter, the parsing operation fails and the method returns `false`.

The `format` parameter is a string that contains either a single standard format specifier or one or more custom format specifiers that define the required pattern of `input`. For details about valid formatting codes, see Standard Date and Time Format Strings and Custom Date and Time Format Strings. If `format` includes the `z`, `zz`, or `zzz` custom format specifiers to indicate that an offset must be present in `input`, that offset must include either a negative sign or a positive sign. If the sign is missing, the parsing operation fails and the method returns `false`.

If `format` requires that `input` contain a date but not a time, the resulting DateTimeOffset object is assigned a time of midnight (0:00:00). If `format` requires that `input` contain a time but not a date, the resulting DateTimeOffset object is assigned the current date on the local system. If `format` does not require that `input` contain an offset, the offset of the resulting DateTimeOffset object depends on the value of the `styles` parameter. If `styles` includes AssumeLocal, the offset of the local time zone is assigned to the DateTimeOffset object. If `styles` includes AssumeUniversal, the Coordinated Universal Time (UTC) offset, or +00:00, is assigned to the DateTimeOffset object. If neither value is specified, the offset of the local time zone is used.

The particular date and time symbols and strings used in `input` are defined by the `formatProvider` parameter. The same is true for the precise pattern of `input` if `format` is a standard format specifier string. The `formatProvider` parameter can be either of the following:

- A CultureInfo object that represents the culture based on which `input` is interpreted. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the symbols and standard formats allowed in `input`.

- A DateTimeFormatInfo object that defines the format of date and time data.

If `formatprovider` is `null`, the CultureInfo object that corresponds to the current culture is used.

The `styles` parameter defines whether white space is allowed in the input string, indicates how strings without an explicit offset component are parsed, and supports UTC conversion as part of the parsing operation. All members of the DateTimeStyles enumeration are supported except NoCurrentDateDefault. The following table lists the effect of each supported member.

Expand table

`DateTimeStyles` member	Behavior
AdjustToUniversal	Parses `input` and, if necessary, converts it to UTC. It is equivalent to parsing a string, and then calling the DateTimeOffset.ToUniversalTime method of the returned DateTimeOffset object.
AssumeLocal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the offset of the local time zone. This is the default behavior.
AssumeUniversal	If `format` does not require that `input` contain an offset value, the returned DateTimeOffset object is given the UTC offset (+00:00).
AllowInnerWhite	Allows `input` to include inner white space not specified by format. Extra white space can appear between date and time components and within individual components, other than the offset, and is ignored when parsing the string.
AllowLeadingWhite	Allows `input` to include leading spaces not specified by `format`. These are ignored when parsing the string.
AllowTrailingWhite	Allows `input` to include trailing spaces not specified by `format`. These are ignored when parsing the string.
AllowWhiteSpaces	Allows `input` to include leading, trailing, and inner spaces not specified by `format`. All extra white-space characters not specified in `format` are ignored when parsing the string.

`DateTimeStyles` member	Behavior
None	Indicates that additional white space is not permitted in `input`. White space must appear exactly as specified in `format`. This is the default behavior.
RoundtripKind	Has no effect, because the DateTimeOffset structure does not include a `Kind` property.

Notes to Callers

In the .NET Framework 4, the TryParseExact returns `false` if the string to be parsed contains an hour component and an AM/PM designator that are not in agreement. In the .NET Framework 3.5 and earlier versions, the AM/PM designator is ignored.

See also

- ParseExact

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

DateTimeOffset.Addition(DateTimeOffset, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds a specified time interval to a DateTimeOffset object that has a specified date and time, and yields a DateTimeOffset object that has new a date and time.

C#

```
public static DateTimeOffset operator + (DateTimeOffset dateTimeOffset,
TimeSpan timeSpan);
```

Parameters

`dateTimeOffset` DateTimeOffset

The object to add the time interval to.

`timeSpan` TimeSpan

The time interval to add.

Returns

DateTimeOffset

An object whose value is the sum of the values of `dateTimeTz` and `timeSpan`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue.

-or-

The resulting DateTimeOffset value is greater than DateTimeOffset.MaxValue.

Remarks

The Addition method defines the addition operation for DateTimeOffset values. It enables code such as the following:

C#

```
DateTimeOffset date1 = new DateTimeOffset(2008, 1, 1, 13, 32, 45,
                       new TimeSpan(-5, 0, 0));
TimeSpan interval1 = new TimeSpan(202, 3, 30, 0);
TimeSpan interval2 = new TimeSpan(5, 0, 0, 0);
DateTimeOffset date2;

Console.WriteLine(date1);         // Displays 1/1/2008 1:32:45 PM -05:00
date2 = date1 + interval1;
Console.WriteLine(date2);         // Displays 7/21/2008 5:02:45 PM -05:00
date2 += interval2;
Console.WriteLine(date2);         // Displays 7/26/2008 5:02:45 PM -05:00
```

Languages that do not support custom operators and operator overloading can call the Add method instead.

The equivalent method for this operator is DateTimeOffset.Add(TimeSpan).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(TimeSpan)
- Performing Arithmetic Operations with Dates and Times

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.Equality(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified DateTimeOffset objects represent the same point in time.

```
C#
public static bool operator == (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if both DateTimeOffset objects have the same UtcDateTime value; otherwise, `false`.

Remarks

The Equality method defines the operation of the equality operator for DateTimeOffset objects. It enables code such as the following:

```
C#
```

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-6, 0, 0));
Console.WriteLine(date1 == date2);        // Displays True
Console.WriteLine(date1 == date3);        // Displays False
```

Before evaluating the `left` and `right` operands for equality, the operator converts both values to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return first.UtcDateTime == second.UtcDateTime;
```

In other words, the Equality method determines whether the two DateTimeOffset objects represent a single point in time. It directly compares neither dates and times nor offsets. To determine whether two DateTimeOffset objects represent the same time and have the same offset value, use the EqualsExact method.

The equivalent method for this operator is DateTimeOffset.Equals(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.GreaterThan(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTimeOffset object is greater than (or later than) a second specified DateTimeOffset object.

C#

```
public static bool operator > (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if the UtcDateTime value of `left` is later than the UtcDateTime value of `right`; otherwise, `false`.

Remarks

The GreaterThan method defines the operation of the greater than operator for DateTimeOffset objects. It enables code such as the following:

C#

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-6, 0, 0));
Console.WriteLine(date1 > date2);        // Displays False
Console.WriteLine(date1 > date3);        // Displays True
```

Languages that do not support custom operators can call the Compare method instead. They can also call the GreaterThan method directly, as the following example shows.

VB

```
Dim date1 As New DateTimeOffset(#6/3/2007 2:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date2 As New DateTimeOffset(#6/3/2007 3:45PM#, _
             New TimeSpan(-6, 0, 0))
Dim date3 As New DateTimeOffset(date1.DateTime, _
             New TimeSpan(-6, 0, 0))
Console.WriteLine(DateTimeOffset.op_GreaterThan(date1, date2))  ' Displays
False
Console.WriteLine(DateTimeOffset.op_GreaterThan(date1, date3))  ' Displays
True
```

Before evaluating the `left` and `right` operands, the operator converts both values to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return left.UtcDateTime > right.UtcDateTime;
```

The equivalent method for this operator is DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.GreaterThanOrEqual(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTimeOffset object is greater than or equal to a second specified DateTimeOffset object.

```
C#

public static bool operator >= (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if the UtcDateTime value of `left` is the same as or later than the UtcDateTime value of `right`; otherwise, `false`.

Remarks

The GreaterThan method defines the operation of the greater than or equal to operator for DateTimeOffset objects. It enables code such as the following:

```
C#
```

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date4 = date1;
Console.WriteLine(date1 >= date2);        // Displays False
Console.WriteLine(date1 >= date3);        // Displays True
Console.WriteLine(date1 >= date4);        // Displays True
```

Languages that do not support custom operators can call the Compare method instead. Some languages can also call the GreaterThanOrEqual method directly, as the following example shows.

VB

```
Dim date1 As New DateTimeOffset(#6/3/2007 2:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date2 As New DateTimeOffset(#6/3/2007 3:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date3 As New DateTimeOffset(date1.DateTime, _
             New TimeSpan(-6, 0, 0))
Dim date4 As DateTimeOffset = date1
Console.WriteLine( _
    DateTimeOffset.op_GreaterThanOrEqual(date1, date2))   ' Displays False
Console.WriteLine( _
    DateTimeOffset.op_GreaterThanOrEqual(date1, date3))   ' Displays True
Console.WriteLine( _
    DateTimeOffset.op_GreaterThanOrEqual(date1, date4))   ' Displays True
```

Before evaluating the `left` and `right` operands, the operator converts both values to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return left.UtcDateTime >= right.UtcDateTime;
```

The equivalent method for this operator is DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Implicit(DateTime to DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an implicit conversion of a DateTime object to a DateTimeOffset object.

C#

```
public static implicit operator DateTimeOffset (DateTime dateTime);
```

Parameters

dateTime DateTime

The object to convert.

Returns

DateTimeOffset

The converted object.

Exceptions

ArgumentOutOfRangeException

The Coordinated Universal Time (UTC) date and time that results from applying the offset is earlier than DateTimeOffset.MinValue.

-or-

The UTC date and time that results from applying the offset is later than DateTimeOffset.MaxValue.

Remarks

The Implicit method enables the compiler to automatically convert a DateTime object to a DateTimeOffset object without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). It defines a widening conversion that does not involve data loss and does not throw an OverflowException. The Implicit method makes code such as the following possible:

C#

```
DateTimeOffset timeWithOffset;
timeWithOffset = new DateTime(2008, 7, 3, 18, 45, 0);
Console.WriteLine(timeWithOffset.ToString());

timeWithOffset = DateTime.UtcNow;
Console.WriteLine(timeWithOffset.ToString());

timeWithOffset = DateTime.SpecifyKind(DateTime.Now,
                                      DateTimeKind.Unspecified);
Console.WriteLine(timeWithOffset.ToString());

timeWithOffset = new DateTime(2008, 7, 1, 2, 30, 0) +
                        new TimeSpan(1, 0, 0, 0);
Console.WriteLine(timeWithOffset.ToString());

timeWithOffset = new DateTime(2008, 1, 1, 2, 30, 0);
Console.WriteLine(timeWithOffset.ToString());
// The example produces the following output if run on 3/20/2007
// at 6:25 PM on a computer in the U.S. Pacific Daylight Time zone:
//       7/3/2008 6:45:00 PM -07:00
//       3/21/2007 1:25:52 AM +00:00
//       3/20/2007 6:25:52 PM -07:00
//       7/2/2008 2:30:00 AM -07:00
//       1/1/2008 2:30:00 AM -08:00
//
// The last example shows automatic adaption to the U.S. Pacific Time
// for winter dates.
```

This method is equivalent to the DateTimeOffset constructor. The offset of the resulting DateTimeOffset object depends on the value of the DateTime.Kind property of the `dateTime` parameter:

- If the value of the DateTime.Kind property is DateTimeKind.Utc, the date and time of the DateTimeOffset object is set equal to `dateTime`, and its Offset property is set equal to 0.
- If the value of the DateTime.Kind property is DateTimeKind.Local or DateTimeKind.Unspecified, the date and time of the DateTimeOffset object is set equal to `dateTime`, and its Offset property is set equal to the offset of the local system's current time zone.

The equivalent method for this operator is DateTimeOffset.DateTimeOffset(DateTime)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Inequality(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified DateTimeOffset objects refer to different points in time.

```
C#
public static bool operator != (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if `left` and `right` do not have the same UtcDateTime value; otherwise, `false`.

Remarks

The Inequality method defines the operation of the inequality operator for DateTimeOffset objects. It always returns the opposite result from Equality. The Inequality method enables code such as the following:

```
C#
```

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-6, 0, 0));
Console.WriteLine(date1 != date2);        // Displays False
Console.WriteLine(date1 != date3);        // Displays True
```

Before evaluating the `left` and `right` operands for equality, the operator converts both values to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return first.UtcDateTime == second.UtcDateTime;
```

In other words, the Inequality method determines whether the two DateTimeOffset objects represent different points in time. It directly compares neither dates and times nor offsets.

Languages that do not support custom operators can call the Compare method instead. In addition, some languages can also call the Inequality method directly, as the following example shows.

VB

```
Dim date1 As New DateTimeOffset(#6/3/2007 2:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date2 As New DateTimeOffset(#6/3/2007 3:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date3 As New DateTimeOffset(date1.DateTime, _
             New TimeSpan(-6, 0, 0))
Dim date4 As DateTimeOffset = date1
Console.WriteLine( _
    DateTimeOffset.op_Inequality(date1, date2))   ' Displays True
Console.WriteLine( _
    DateTimeOffset.op_Inequality(date1, date3))   ' Displays True
Console.WriteLine( _
    DateTimeOffset.op_Inequality(date1, date4))   ' Displays False
```

The equivalent method for this operator is DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.LessThan(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTimeOffset object is less than a second specified DateTimeOffset object.

```
C#
public static bool operator < (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if the UtcDateTime value of `left` is earlier than the UtcDateTime value of `right`; otherwise, `false`.

Remarks

The LessThan method defines the operation of the less than operator for DateTimeOffset objects. It enables code such as the following:

```
C#
```

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-8, 0, 0));
Console.WriteLine(date1 < date2);        // Displays False
Console.WriteLine(date1 < date3);        // Displays True
```

Before evaluating the `left` and `right` operands, the operator converts both operands to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return left.UtcDateTime < right.UtcDateTime;
```

Languages that do not support custom operators can call the Compare method instead. In addition, some languages can also call the LessThan method directly, as the following example shows.

VB

```
Dim date1 As New DateTimeOffset(#6/3/2007 2:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date2 As New DateTimeOffset(#6/3/2007 3:45PM#, _
             New TimeSpan(-6, 0, 0))
Dim date3 As New DateTimeOffset(date1.DateTime, _
             New TimeSpan(-8, 0, 0))
Console.WriteLine(DateTimeOffset.op_LessThan(date1, date2))  ' Displays
False
Console.WriteLine(DateTimeOffset.op_LessThan(date1, date3))  ' Displays True
```

The equivalent method for this operator is DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.LessThanOrEqual(DateTimeOffset, DateTimeOffset) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified DateTimeOffset object is less than a second specified DateTimeOffset object.

C#

```
public static bool operator <= (DateTimeOffset left, DateTimeOffset right);
```

Parameters

`left` DateTimeOffset

The first object to compare.

`right` DateTimeOffset

The second object to compare.

Returns

Boolean

`true` if the UtcDateTime value of `left` is earlier than the UtcDateTime value of `right`; otherwise, `false`.

Remarks

The LessThanOrEqual method defines the operation of the less than or equal to operator for DateTimeOffset objects. It enables code such as the following:

C#

```
DateTimeOffset date1 = new DateTimeOffset(2007, 6, 3, 14, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date2 = new DateTimeOffset(2007, 6, 3, 15, 45, 0,
             new TimeSpan(-7, 0, 0));
DateTimeOffset date3 = new DateTimeOffset(date1.DateTime,
             new TimeSpan(-6, 0, 0));
DateTimeOffset date4 = date1;
Console.WriteLine(date1 <= date2);        // Displays True
Console.WriteLine(date1 <= date3);        // Displays False
Console.WriteLine(date1 <= date4);        // Displays True
```

Languages that do not support custom operators can call the Compare method instead. Some languages can also call the LessThanOrEqual method directly, as the following example shows.

VB

```
Dim date1 As New DateTimeOffset(#6/3/2007 2:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date2 As New DateTimeOffset(#6/3/2007 3:45PM#, _
             New TimeSpan(-7, 0, 0))
Dim date3 As New DateTimeOffset(date1.DateTime, _
             New TimeSpan(-6, 0, 0))
Dim date4 As DateTimeOffset = date1
Console.WriteLine(DateTimeOffset.op_LessThanOrEqual(date1, date2))   '
Displays True
Console.WriteLine(DateTimeOffset.op_LessThanOrEqual(date1, date3)) '
Displays False
Console.WriteLine(DateTimeOffset.op_LessThanOrEqual(date1, date4)) '
Displays True
```

Before evaluating the `left` and `right` operands, the operator converts both values to Coordinated Universal Time (UTC). The operation is equivalent to the following:

C#

```
return left.UtcDateTime <= right.UtcDateTime;
```

The equivalent method for this operator is DateTimeOffset.Compare(DateTimeOffset, DateTimeOffset)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts a specified DateTimeOffset or TimeSpan object from a DateTimeOffset object.

Overloads

Expand table

Subtraction(DateTimeOffset, DateTimeOffset)	Subtracts one DateTimeOffset object from another and yields a time interval.
Subtraction(DateTimeOffset, TimeSpan)	Subtracts a specified time interval from a specified date and time, and yields a new date and time.

Subtraction(DateTimeOffset, DateTimeOffset)

Subtracts one DateTimeOffset object from another and yields a time interval.

C#

```
public static TimeSpan operator - (DateTimeOffset left, DateTimeOffset
right);
```

Parameters

`left` DateTimeOffset

The minuend.

`right` DateTimeOffset

The subtrahend.

Returns

TimeSpan

An object that represents the difference between `left` and `right`.

Remarks

The Subtraction method defines the subtraction operation for DateTimeOffset objects. It enables code such as the following:

C#

```
DateTimeOffset firstDate = new DateTimeOffset(2008, 3, 25, 18, 0, 0,
                                              new TimeSpan(-7, 0, 0));
DateTimeOffset secondDate = new DateTimeOffset(2008, 3, 25, 18, 0, 0,
                                               new TimeSpan(-5, 0, 0));
DateTimeOffset thirdDate = new DateTimeOffset(2008, 2, 28, 9, 0, 0,
                                              new TimeSpan(-7, 0, 0));
TimeSpan difference;

difference = firstDate - secondDate;
Console.WriteLine("({0}) - ({1}): {2} days, {3}:{4:d2}",
                  firstDate.ToString(),
                  secondDate.ToString(),
                  difference.Days,
                  difference.Hours,
                  difference.Minutes);

difference = firstDate - thirdDate;
Console.WriteLine("({0}) - ({1}): {2} days, {3}:{4:d2}",
                  firstDate.ToString(),
                  secondDate.ToString(),
                  difference.Days,
                  difference.Hours,
                  difference.Minutes);
// The example produces the following output:
//    (3/25/2008 6:00:00 PM -07:00) - (3/25/2008 6:00:00 PM -05:00): 0
days, 2:00
//    (3/25/2008 6:00:00 PM -07:00) - (3/25/2008 6:00:00 PM -05:00): 26
days, 9:00
```

Languages that do not support custom operators and operator overloading can call the DateTimeOffset.Subtract(DateTimeOffset) method instead.

The equivalent method for this operator is DateTimeOffset.Subtract(TimeSpan).

See also

- Subtract(DateTimeOffset)

- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Subtraction(DateTimeOffset, TimeSpan)

Subtracts a specified time interval from a specified date and time, and yields a new date and time.

C#

```
public static DateTimeOffset operator - (DateTimeOffset dateTimeOffset,
TimeSpan timeSpan);
```

Parameters

`dateTimeOffset` DateTimeOffset

The date and time object to subtract from.

`timeSpan` TimeSpan

The time interval to subtract.

Returns

DateTimeOffset

An object that is equal to the value of `dateTimeOffset` minus `timeSpan`.

Exceptions

ArgumentOutOfRangeException

The resulting DateTimeOffset value is less than DateTimeOffset.MinValue or greater than DateTimeOffset.MaxValue.

Remarks

The Subtraction method defines the subtraction operation for DateTimeOffset objects. It enables code such as the following:

```
C#

DateTimeOffset offsetDate = new DateTimeOffset(2007, 12, 3, 11, 30, 0,
                                new TimeSpan(-8, 0, 0));
TimeSpan duration = new TimeSpan(7, 18, 0, 0);
Console.WriteLine(offsetDate - duration);  // Displays 11/25/2007 5:30:00
PM -08:00
```

Languages that do not support custom operators and operator overloading can call the DateTimeOffset.Subtract(TimeSpan) method instead.

The equivalent method for this operator is DateTimeOffset.Subtract(TimeSpan).

See also

- Subtract(TimeSpan)
- Performing Arithmetic Operations with Dates and Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the value of the current DateTimeOffset object with another object of the same type.

```
C#

int IComparable.CompareTo (object obj);
```

Parameters

obj Object

The object to compare with the current DateTimeOffset object.

Returns

Int32

A 32-bit signed integer that indicates whether the current DateTimeOffset object is less than, equal to, or greater than `obj`. The return values of the method are interpreted as follows:

Expand table

Return Value	Description
Less than zero	The current DateTimeOffset object is less than (earlier than) `obj`.
Zero	The current DateTimeOffset object is equal to (the same point in time as) `obj`.
Greater than zero	The current DateTimeOffset object is greater than (later than) `obj`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DateTimeOffset.IDeserialization Callback.OnDeserialization(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Runs when the deserialization of an object has been completed.

```
C#
void IDeserializationCallback.OnDeserialization (object sender);
```

Parameters

sender Object

The object that initiated the callback. The functionality for this parameter is not currently implemented.

Implements

OnDeserialization(Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DateTimeOffset.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data required to serialize the current DateTimeOffset object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with data.

`context` StreamingContext

The destination for this serialization (see StreamingContext).

Implements

GetObjectData(SerializationInfo, StreamingContext)

Exceptions

ArgumentNullException

The `info` parameter is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DayOfWeek Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the day of the week.

C#

```
public enum DayOfWeek
```

Inheritance Object → ValueType → Enum → DayOfWeek

Fields

Expand table

Friday	5	Indicates Friday.
Monday	1	Indicates Monday.
Saturday	6	Indicates Saturday.
Sunday	0	Indicates Sunday.
Thursday	4	Indicates Thursday.
Tuesday	2	Indicates Tuesday.
Wednesday	3	Indicates Wednesday.

Examples

The following example demonstrates the DateTime.DayOfWeek property and the DayOfWeek enumeration.

C#

```
// This example demonstrates the DateTime.DayOfWeek property
using System;

class Sample
{
    public static void Main()
    {
// Assume the current culture is en-US.
// Create a DateTime for the first of May, 2003.
    DateTime dt = new DateTime(2003, 5, 1);
    Console.WriteLine("Is Thursday the day of the week for {0:d}?: {1}",
                      dt, dt.DayOfWeek == DayOfWeek.Thursday);
    Console.WriteLine("The day of the week for {0:d} is {1}.", dt,
dt.DayOfWeek);
    }
}
/*
This example produces the following results:

Is Thursday the day of the week for 5/1/2003?: True
The day of the week for 5/1/2003 is Thursday.
*/
```

Remarks

The DayOfWeek enumeration represents the day of the week in calendars that have seven days per week. The value of the constants in this enumeration ranges from Sunday to Saturday. If cast to an integer, its value ranges from zero (which indicates Sunday) to six (which indicates Saturday).

This enumeration is useful when it is desirable to have a strongly typed specification of the day of the week. For example, this enumeration is the type of the property value for the DateTime.DayOfWeek and DateTimeOffset.DayOfWeek properties.

The members of the DayOfWeek enumeration are not localized. To return the localized name of the day of the week, call the DateTime.ToString(String) or the DateTime.ToString(String, IFormatProvider) method with either the "ddd" or "dddd" format strings. The former format string produces the abbreviated weekday name; the latter produces the full weekday name.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a nonexistent value. This class cannot be inherited.

C#

```
public sealed class DBNull : IConvertible,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → DBNull

Implements IConvertible , ISerializable

Examples

The following example calls the `DBNull.Value.Equals` method to determine whether a database field in a contacts database has a valid value. If it does, the field value is appended to the string output in a label.

C#

```
private void OutputLabels(DataTable dt)
{
   string label;

   // Iterate rows of table
   foreach (DataRow row in dt.Rows)
   {
      int labelLen;
      label = String.Empty;
      label += AddFieldValue(label, row, "Title");
      label += AddFieldValue(label, row, "FirstName");
      label += AddFieldValue(label, row, "MiddleInitial");
      label += AddFieldValue(label, row, "LastName");
      label += AddFieldValue(label, row, "Suffix");
      label += "\n";
      label += AddFieldValue(label, row, "Address1");
      label += AddFieldValue(label, row, "AptNo");
```

```
        label += "\n";
        labelLen = label.Length;
        label += AddFieldValue(label, row, "Address2");
        if (label.Length != labelLen)
           label += "\n";
        label += AddFieldValue(label, row, "City");
        label += AddFieldValue(label, row, "State");
        label += AddFieldValue(label, row, "Zip");
        Console.WriteLine(label);
        Console.WriteLine();
    }
}

private string AddFieldValue(string label, DataRow row,
                             string fieldName)
{
    if (! DBNull.Value.Equals(row[fieldName]))
        return (string) row[fieldName] + " ";
    else
        return String.Empty;
}
```

Remarks

The DBNull class represents a nonexistent value. In a database, for example, a column in a row of a table might not contain any data whatsoever. That is, the column is considered to not exist at all instead of merely not having a value. A DBNull object represents the nonexistent column. Additionally, COM interop uses the DBNull class to distinguish between a VT_NULL variant, which indicates a nonexistent value, and a VT_EMPTY variant, which indicates an unspecified value.

The DBNull type is a singleton class, which means only one DBNull object exists. The DBNull.Value member represents the sole DBNull object. DBNull.Value can be used to explicitly assign a nonexistent value to a database field, although most ADO.NET data providers automatically assign values of DBNull when a field does not have a valid value. You can determine whether a value retrieved from a database field is a DBNull value by passing the value of that field to the `DBNull.Value.Equals` method. However, some languages and database objects supply methods that make it easier to determine whether the value of a database field is DBNull.Value. These include the Visual Basic `IsDBNull` function, the Convert.IsDBNull method, the DataTableReader.IsDBNull method, and the IDataRecord.IsDBNull method.

Do not confuse the notion of `null` in an object-oriented programming language with a DBNull object. In an object-oriented programming language, `null` means the absence

of a reference to an object. DBNull represents an uninitialized variant or nonexistent database column.

Fields

Expand table

Value	Represents the sole instance of the DBNull class.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Implements the ISerializable interface and returns the data needed to serialize the DBNull object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetTypeCode()	Gets the TypeCode value for DBNull.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns an empty string (Empty).
ToString(IFormatProvider)	Returns an empty string using the specified IFormatProvider.

Explicit Interface Implementations

Expand table

IConvertible.ToBoolean(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

IConvertible.ToByte(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToDouble(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToInt16(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToInt32(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToInt64(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToSByte(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToSingle(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToType(Type, IFormatProvider)	Converts the current DBNull object to the specified type.
IConvertible.ToUInt16(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToUInt32(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.
IConvertible.ToUInt64(IFormatProvider)	This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

See also

- DBNull
- IsDBNull(Object)
- VarEnum

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.Value Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the sole instance of the DBNull class.

C#

```
public static readonly DBNull Value;
```

Field Value

DBNull

Examples

The following example calls the `DBNull.Value.Equals` method to determine whether a database field in a contacts database has a valid value. If it does, the field value is appended to the string output in a label.

C#

```
private void OutputLabels(DataTable dt)
{
   string label;

   // Iterate rows of table
   foreach (DataRow row in dt.Rows)
   {
      int labelLen;
      label = String.Empty;
      label += AddFieldValue(label, row, "Title");
      label += AddFieldValue(label, row, "FirstName");
      label += AddFieldValue(label, row, "MiddleInitial");
      label += AddFieldValue(label, row, "LastName");
      label += AddFieldValue(label, row, "Suffix");
      label += "\n";
      label += AddFieldValue(label, row, "Address1");
      label += AddFieldValue(label, row, "AptNo");
      label += "\n";
```

```
        labelLen = label.Length;
        label += AddFieldValue(label, row, "Address2");
        if (label.Length != labelLen)
           label += "\n";
        label += AddFieldValue(label, row, "City");
        label += AddFieldValue(label, row, "State");
        label += AddFieldValue(label, row, "Zip");
        Console.WriteLine(label);
        Console.WriteLine();
    }
}

private string AddFieldValue(string label, DataRow row,
                             string fieldName)
{
    if (! DBNull.Value.Equals(row[fieldName]))
        return (string) row[fieldName] + " ";
    else
        return String.Empty;
}
```

Remarks

DBNull is a singleton class, which means only this instance of this class can exist.

If a database field has missing data, you can use the DBNull.Value property to explicitly assign a DBNull object value to the field. However, most data providers do this automatically.

To evaluate database fields to determine whether their values are DBNull, you can pass the field value to the `DBNull.Value.Equals` method. However, this method is rarely used because there are a number of other ways to evaluate a database field for missing data. These include the Visual Basic `IsDBNull` function, the Convert.IsDBNull method, the DataTableReader.IsDBNull method, the IDataRecord.IsDBNull method, and several other methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DBNull.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Implements the ISerializable interface and returns the data needed to serialize the DBNull object.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

A SerializationInfo object containing information required to serialize the DBNull object.

`context` StreamingContext

A StreamingContext object containing the source and destination of the serialized stream associated with the DBNull object.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ISerializable
- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the TypeCode value for DBNull.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The TypeCode value for DBNull, which is DBNull.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DBNull.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an empty string.

Overloads

Expand table

ToString()	Returns an empty string (Empty).
ToString(IFormatProvider)	Returns an empty string using the specified IFormatProvider.

ToString()

Returns an empty string (Empty).

C#

```
public override string ToString ();
```

Returns

String

An empty string (Empty).

See also

- Empty

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Returns an empty string using the specified IFormatProvider.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

The IFormatProvider to be used to format the return value.

-or-

`null` to obtain the format information from the current locale setting of the operating system.

Returns

String

An empty string (Empty).

Implements

ToString(IFormatProvider)

See also

- Empty

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Boolean

None. The return value for this member is not used.

Implements

ToBoolean(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Byte

None. The return value for this member is not used.

Implements

ToByte(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Char

None. The return value for this member is not used.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

DateTime

None. The return value for this member is not used.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Decimal

None. The return value for this member is not used.

Implements

ToDecimal(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Double

None. The return value for this member is not used.

Implements

ToDouble(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int16

None. The return value for this member is not used.

Implements

ToInt16(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int32

None. The return value for this member is not used.

Implements

ToInt32(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Int64

None. The return value for this member is not used.

Implements

ToInt64(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

SByte

None. The return value for this member is not used.

Implements

ToSByte(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

Single

None. The return value for this member is not used.

Implements

ToSingle(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current DBNull object to the specified type.

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to convert the current DBNull object to.

provider IFormatProvider

An object that implements the IFormatProvider interface and is used to augment the conversion. If `null` is specified, format information is obtained from the current culture.

Returns

Object

The boxed equivalent of the current DBNull object, if that conversion is supported; otherwise, an exception is thrown and no value is returned.

Implements

ToType(Type, IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

ArgumentNullException

`type` is `null`.

Remarks

Three conversions are supported: If the `type` parameter specifies Object or DBNull, the return value is the current DBNull object itself. If the `type` parameter specifies String, the return value is the string returned by the ToString method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Type
- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt16

None. The return value for this member is not used.

Implements

ToUInt16(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#

uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt32

None. The return value for this member is not used.

Implements

ToUInt32(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DBNull.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

This conversion is not supported. Attempting to make this conversion throws an InvalidCastException.

```
C#
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that implements the IFormatProvider interface. (This parameter is not used; specify `null`.)

Returns

UInt64

None. The return value for this member is not used.

Implements

ToUInt64(IFormatProvider)

Exceptions

InvalidCastException

This conversion is not supported for the DBNull type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IConvertible

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a decimal floating-point number.

C#

```
public readonly struct Decimal : IComparable<decimal>, IConvertible,
IEquatable<decimal>, IParsable<decimal>, ISpanParsable<decimal>,
IUtf8SpanParsable<decimal>,
System.Numerics.IAdditionOperators<decimal,decimal,decimal>,
System.Numerics.IAdditiveIdentity<decimal,decimal>,
System.Numerics.IComparisonOperators<decimal,decimal,bool>,
System.Numerics.IDecrementOperators<decimal>,
System.Numerics.IDivisionOperators<decimal,decimal,decimal>,
System.Numerics.IEqualityOperators<decimal,decimal,bool>,
System.Numerics.IFloatingPoint<decimal>,
System.Numerics.IFloatingPointConstants<decimal>,
System.Numerics.IIncrementOperators<decimal>,
System.Numerics.IMinMaxValue<decimal>,
System.Numerics.IModulusOperators<decimal,decimal,decimal>,
System.Numerics.IMultiplicativeIdentity<decimal,decimal>,
System.Numerics.IMultiplyOperators<decimal,decimal,decimal>,
System.Numerics.INumber<decimal>, System.Numerics.INumberBase<decimal>,
System.Numerics.ISignedNumber<decimal>,
System.Numerics.ISubtractionOperators<decimal,decimal,decimal>,
System.Numerics.IUnaryNegationOperators<decimal,decimal>,
System.Numerics.IUnaryPlusOperators<decimal,decimal>,
System.Runtime.Serialization.IDeserializationCallback,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → Decimal

Implements IComparable , IComparable<Decimal> , IConvertible , IEquatable<Decimal> , IFormattable , IDeserializationCallback , ISerializable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Decimal> , IParsable<TSelf> , ISpanParsable<Decimal> , ISpanParsable<TSelf> , IAdditionOperators<Decimal,Decimal,Decimal> ,

IAdditionOperators<TSelf,TSelf,TSelf> ,

IAdditiveIdentity<Decimal,Decimal> , IAdditiveIdentity<TSelf,TSelf> ,

IComparisonOperators<Decimal,Decimal,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<Decimal> , IDecrementOperators<TSelf> ,

IDivisionOperators<Decimal,Decimal,Decimal> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<Decimal,Decimal,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IFloatingPoint<Decimal> ,

IFloatingPointConstants<Decimal> , IFloatingPointConstants<TSelf> ,

IIncrementOperators<Decimal> , IIncrementOperators<TSelf> ,

IMinMaxValue<Decimal> ,

IModulusOperators<Decimal,Decimal,Decimal> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<Decimal,Decimal> ,

IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<Decimal,Decimal,Decimal> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Decimal> ,

INumber<TSelf> , INumberBase<Decimal> , INumberBase<TSelf> ,

ISignedNumber<Decimal> , ISignedNumber<TSelf> ,

ISubtractionOperators<Decimal,Decimal,Decimal> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<Decimal,Decimal> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Decimal,Decimal> ,

IUnaryPlusOperators<TSelf,TSelf> , IUtf8SpanFormattable ,

IUtf8SpanParsable<Decimal> , IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Decimal.

Constructors

Expand table

Decimal(Double)	Initializes a new instance of Decimal to the value of the specified double-precision floating-point number.
Decimal(Int32)	Initializes a new instance of Decimal to the value of the specified 32-bit signed integer.
Decimal(Int32, Int32, Int32, Boolean, Byte)	Initializes a new instance of Decimal from parameters specifying the instance's constituent parts.
Decimal(Int32[])	Initializes a new instance of Decimal to a decimal value represented in binary and contained in a specified array.
Decimal(Int64)	Initializes a new instance of Decimal to the value of the specified 64-bit signed integer.
Decimal(ReadOnlySpan<Int32>)	Initializes a new instance of Decimal to a decimal value represented in binary and contained in the specified span.
Decimal(Single)	Initializes a new instance of Decimal to the value of the specified single-precision floating-point number.
Decimal(UInt32)	Initializes a new instance of Decimal to the value of the specified 32-bit unsigned integer.
Decimal(UInt64)	Initializes a new instance of Decimal to the value of the specified 64-bit unsigned integer.

Fields

Expand table

MaxValue	Represents the largest possible value of Decimal. This field is constant and read-only.
MinusOne	Represents the number negative one (-1).
MinValue	Represents the smallest possible value of Decimal. This field is constant and read-only.
One	Represents the number one (1).
Zero	Represents the number zero (0).

Properties

Expand table

Scale	Gets the scaling factor of the decimal, which is a number from 0 to 28 that represents the number of decimal digits.

Methods

Expand table

Abs(Decimal)	Computes the absolute of a value.
Add(Decimal, Decimal)	Adds two specified Decimal values.
Ceiling(Decimal)	Returns the smallest integral value that is greater than or equal to the specified decimal number.
Clamp(Decimal, Decimal, Decimal)	Clamps a value to an inclusive minimum and maximum value.
Compare(Decimal, Decimal)	Compares two specified Decimal values.
CompareTo(Decimal)	Compares this instance to a specified Decimal object and returns a comparison of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns a comparison of their relative values.
CopySign(Decimal, Decimal)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
Divide(Decimal, Decimal)	Divides two specified Decimal values.
Equals(Decimal)	Returns a value indicating whether this instance and a specified Decimal object represent the same value.

Equals(Decimal, Decimal)	Returns a value indicating whether two specified instances of Decimal represent the same value.
Equals(Object)	Returns a value indicating whether this instance and a specified Object represent the same type and value.
Floor(Decimal)	Rounds a specified Decimal number to the closest integer toward negative infinity.
FromOACurrency(Int64)	Converts the specified 64-bit signed integer, which contains an OLE Automation Currency value, to the equivalent Decimal value.
GetBits(Decimal)	Converts the value of a specified instance of Decimal to its equivalent binary representation.
GetBits(Decimal, Span<Int32>)	Converts the value of a specified instance of Decimal to its equivalent binary representation.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Decimal.
IsCanonical(Decimal)	Determines if a value is in its canonical representation.
IsEvenInteger(Decimal)	Determines if a value represents an even integral number.
IsInteger(Decimal)	Determines if a value represents an integral number.
IsNegative(Decimal)	Determines if a value is negative.
IsOddInteger(Decimal)	Determines if a value represents an odd integral number.
IsPositive(Decimal)	Determines if a value is positive.
Max(Decimal, Decimal)	Compares two values to compute which is greater.
MaxMagnitude(Decimal, Decimal)	Compares two values to compute which is greater.
Min(Decimal, Decimal)	Compares two values to compute which is lesser.
MinMagnitude(Decimal, Decimal)	Compares two values to compute which is lesser.
Multiply(Decimal, Decimal)	Multiplies two specified Decimal values.
Negate(Decimal)	Returns the result of multiplying the specified Decimal value by negative one.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.

Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format.
Parse(String)	Converts the string representation of a number to its Decimal equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number to its Decimal equivalent using the specified culture-specific format information.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its Decimal equivalent.
Parse(String, NumberStyles, IFormat Provider)	Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format.
Remainder(Decimal, Decimal)	Computes the remainder after dividing two Decimal values.
Round(Decimal)	Rounds a decimal value to the nearest integer.
Round(Decimal, Int32)	Rounds a Decimal value to a specified number of decimal places.
Round(Decimal, Int32, Midpoint Rounding)	Rounds a decimal value to the specified precision using the specified rounding strategy.
Round(Decimal, MidpointRounding)	Rounds a decimal value to an integer using the specified rounding strategy.
Sign(Decimal)	Computes the sign of a value.
Subtract(Decimal, Decimal)	Subtracts one specified Decimal value from another.
ToByte(Decimal)	Converts the value of the specified Decimal to the equivalent 8-bit unsigned integer.
ToDouble(Decimal)	Converts the value of the specified Decimal to the equivalent double-precision floating-point number.
ToInt16(Decimal)	Converts the value of the specified Decimal to the equivalent 16-bit signed integer.
ToInt32(Decimal)	Converts the value of the specified Decimal to the equivalent 32-bit signed integer.

ToInt64(Decimal)	Converts the value of the specified Decimal to the equivalent 64-bit signed integer.
ToOACurrency(Decimal)	Converts the specified Decimal value to the equivalent OLE Automation Currency value, which is contained in a 64-bit signed integer.
ToSByte(Decimal)	Converts the value of the specified Decimal to the equivalent 8-bit signed integer.
ToSingle(Decimal)	Converts the value of the specified Decimal to the equivalent single-precision floating-point number.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToUInt16(Decimal)	Converts the value of the specified Decimal to the equivalent 16-bit unsigned integer.
ToUInt32(Decimal)	Converts the value of the specified Decimal to the equivalent 32-bit unsigned integer.
ToUInt64(Decimal)	Converts the value of the specified Decimal to the equivalent 64-bit unsigned integer.
Truncate(Decimal)	Returns the integral digits of the specified Decimal; any fractional digits are discarded.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current decimal instance into the provided span of characters.
TryGetBits(Decimal, Span<Int32>, Int32)	Tries to convert the value of a specified instance of Decimal to its equivalent binary representation.
TryParse(ReadOnlySpan<Byte>, Decimal)	Tries to convert a UTF-8 character span containing the string representation of a number to its signed decimal equivalent.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Decimal)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Decimal)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Decimal)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Decimal)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Decimal)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Decimal)	Converts the string representation of a number to its Decimal equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Decimal)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Decimal)	Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

Operators

Expand table

Addition(Decimal, Decimal)	Adds two specified Decimal values.
Decrement(Decimal)	Decrements the Decimal operand by one.
Division(Decimal, Decimal)	Divides two specified Decimal values.
Equality(Decimal, Decimal)	Returns a value that indicates whether two Decimal values are equal.
Explicit(Decimal to Byte)	Defines an explicit conversion of a Decimal to an 8-bit unsigned integer.

Explicit(Decimal to Char)	Defines an explicit conversion of a Decimal to a Unicode character.
Explicit(Decimal to Double)	Defines an explicit conversion of a Decimal to a double-precision floating-point number.
Explicit(Decimal to Int16)	Defines an explicit conversion of a Decimal to a 16-bit signed integer.
Explicit(Decimal to Int32)	Defines an explicit conversion of a Decimal to a 32-bit signed integer.
Explicit(Decimal to Int64)	Defines an explicit conversion of a Decimal to a 64-bit signed integer.
Explicit(Decimal to SByte)	Defines an explicit conversion of a Decimal to an 8-bit signed integer. This API is not CLS-compliant.
Explicit(Decimal to Single)	Defines an explicit conversion of a Decimal to a single-precision floating-point number.
Explicit(Decimal to UInt16)	Defines an explicit conversion of a Decimal to a 16-bit unsigned integer. This API is not CLS-compliant.
Explicit(Decimal to UInt32)	Defines an explicit conversion of a Decimal to a 32-bit unsigned integer. This API is not CLS-compliant.
Explicit(Decimal to UInt64)	Defines an explicit conversion of a Decimal to a 64-bit unsigned integer. This API is not CLS-compliant.
Explicit(Double to Decimal)	Defines an explicit conversion of a double-precision floating-point number to a Decimal.
Explicit(Single to Decimal)	Defines an explicit conversion of a single-precision floating-point number to a Decimal.
GreaterThan(Decimal, Decimal)	Returns a value indicating whether a specified Decimal is greater than another specified Decimal.
GreaterThanOrEqual(Decimal, Decimal)	Returns a value indicating whether a specified Decimal is greater than or equal to another specified Decimal.
Implicit(Byte to Decimal)	Defines an implicit conversion of an 8-bit unsigned integer to a Decimal.

Implicit(Char to Decimal)	Defines an implicit conversion of a Unicode character to a Decimal.
Implicit(Int16 to Decimal)	Defines an implicit conversion of a 16-bit signed integer to a Decimal.
Implicit(Int32 to Decimal)	Defines an implicit conversion of a 32-bit signed integer to a Decimal.
Implicit(Int64 to Decimal)	Defines an implicit conversion of a 64-bit signed integer to a Decimal.
Implicit(SByte to Decimal)	Defines an implicit conversion of an 8-bit signed integer to a Decimal. This API is not CLS-compliant.
Implicit(UInt16 to Decimal)	Defines an implicit conversion of a 16-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Implicit(UInt32 to Decimal)	Defines an implicit conversion of a 32-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Implicit(UInt64 to Decimal)	Defines an implicit conversion of a 64-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Increment(Decimal)	Increments the Decimal operand by 1.
Inequality(Decimal, Decimal)	Returns a value that indicates whether two Decimal objects have different values.
LessThan(Decimal, Decimal)	Returns a value indicating whether a specified Decimal is less than another specified Decimal.
LessThanOrEqual(Decimal, Decimal)	Returns a value indicating whether a specified Decimal is less than or equal to another specified Decimal.
Modulus(Decimal, Decimal)	Returns the remainder resulting from dividing two specified Decimal values.
Multiply(Decimal, Decimal)	Multiplies two specified Decimal values.
Subtraction(Decimal, Decimal)	Subtracts two specified Decimal values.
UnaryNegation(Decimal)	Negates the value of the specified Decimal operand.

UnaryPlus(Decimal)	Returns the value of the Decimal operand (the sign of the operand is unchanged).

Explicit Interface Implementations

Expand table

IAdditiveIdentity<Decimal,Decimal>.AdditiveIdentity	Gets the additive identity of the current type.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).

IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IDeserializationCallback.OnDeserialization(Object)	Runs when the deserialization of an object has been completed.
IFloatingPoint<Decimal>.GetExponentByteCount()	Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Decimal>.GetExponentShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current exponent.
IFloatingPoint<Decimal>.GetSignificandBitLength()	Gets the length, in bits, of the current significand.
IFloatingPoint<Decimal>.GetSignificandByteCount()	Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Decimal>.TryWriteExponentBigEndian(Span<Byte>, Int32)	Tries to write the current exponent, in big-endian format, to a given span.
IFloatingPoint<Decimal>.TryWriteExponentLittleEndian(Span<Byte>, Int32)	Tries to write the current exponent, in little-endian format, to a given span.
IFloatingPoint<Decimal>.TryWriteSignificandBigEndian(Span<Byte>, Int32)	Tries to write the current significand, in big-endian format, to a given span.
IFloatingPoint<Decimal>.TryWriteSignificandLittleEndian(Span<Byte>, Int32)	Tries to write the current significand, in little-endian format, to a given span.
IFloatingPointConstants<Decimal>.E	Gets the mathematical constant `e`.
IFloatingPointConstants<Decimal>.Pi	Gets the mathematical constant `pi`.
IFloatingPointConstants<Decimal>.Tau	Gets the mathematical constant `tau`.
IMinMaxValue<Decimal>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Decimal>.MinValue	Gets the minimum value of the current type.

IMultiplicativeIdentity<Decimal,Decimal>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
INumber<Decimal>.MaxNumber(Decimal, Decimal)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<Decimal>.MinNumber(Decimal, Decimal)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Decimal>.IsComplexNumber(Decimal)	Determines if a value represents a complex number.
INumberBase<Decimal>.IsFinite(Decimal)	Determines if a value is finite.
INumberBase<Decimal>.IsImaginaryNumber(Decimal)	Determines if a value represents a pure imaginary number.
INumberBase<Decimal>.IsInfinity(Decimal)	Determines if a value is infinite.
INumberBase<Decimal>.IsNaN(Decimal)	Determines if a value is NaN.
INumberBase<Decimal>.IsNegativeInfinity(Decimal)	Determines if a value is negative infinity.
INumberBase<Decimal>.IsNormal(Decimal)	Determines if a value is normal.
INumberBase<Decimal>.IsPositiveInfinity(Decimal)	Determines if a value is positive infinity.
INumberBase<Decimal>.IsRealNumber(Decimal)	Determines if a value represents a real number.
INumberBase<Decimal>.IsSubnormal(Decimal)	Determines if a value is subnormal.
INumberBase<Decimal>.IsZero(Decimal)	Determines if a value is zero.
INumberBase<Decimal>.MaxMagnitudeNumber(Decimal, Decimal)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Decimal>.MinMagnitudeNumber(Decimal, Decimal)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Decimal>.One	Gets the value `1` for the type.
INumberBase<Decimal>.Radix	Gets the radix, or base, for the type.
INumberBase<Decimal>.TryConvertFromChecked<TOther>(TOther, Decimal)	

INumberBase<Decimal>.TryConvertFromSaturating<TOther>(TOther, Decimal)	
INumberBase<Decimal>.TryConvertFromTruncating<TOther>(TOther, Decimal)	
INumberBase<Decimal>.TryConvertToChecked<TOther>(Decimal, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Decimal>.TryConvertToSaturating<TOther>(Decimal, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Decimal>.TryConvertToTruncating<TOther>(Decimal, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Decimal>.Zero	Gets the value `0` for the type.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo with the data needed to serialize the target object.
ISignedNumber<Decimal>.NegativeOne	Gets the value `-1` for the type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- SByte
- Int16
- Int32
- Int64
- Byte
- UInt16
- UInt32
- UInt64
- Single
- Double
- Char

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of Decimal.

Overloads

Expand table

Decimal(Double)	Initializes a new instance of Decimal to the value of the specified double-precision floating-point number.
Decimal(Int32)	Initializes a new instance of Decimal to the value of the specified 32-bit signed integer.
Decimal(Int32[])	Initializes a new instance of Decimal to a decimal value represented in binary and contained in a specified array.
Decimal(Int64)	Initializes a new instance of Decimal to the value of the specified 64-bit signed integer.
Decimal(ReadOnlySpan<Int32>)	Initializes a new instance of Decimal to a decimal value represented in binary and contained in the specified span.
Decimal(Single)	Initializes a new instance of Decimal to the value of the specified single-precision floating-point number.
Decimal(UInt32)	Initializes a new instance of Decimal to the value of the specified 32-bit unsigned integer.
Decimal(UInt64)	Initializes a new instance of Decimal to the value of the specified 64-bit unsigned integer.
Decimal(Int32, Int32, Int32, Boolean, Byte)	Initializes a new instance of Decimal from parameters specifying the instance's constituent parts.

Decimal(Double)

Initializes a new instance of Decimal to the value of the specified double-precision floating-point number.

```
C#

public Decimal (double value);
```

Parameters

value Double

The value to represent as a Decimal.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

-or-

`value` is NaN, PositiveInfinity, or NegativeInfinity.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with a Double value.

```
C#

// Example of the decimal( double ) constructor.
using System;

class DecimalCtorDoDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' )+1 );
    }

    // Create a decimal object and display its value.
    public static void CreateDecimal( double value, string valToStr )
    {
        // Format and display the constructor.
        Console.Write( "{0,-34}",
            String.Format( "decimal( {0} )", valToStr ) );
```

```
        try
        {
            // Construct the decimal value.
            decimal decimalNum = new decimal( value );

            // Display the value if it was created successfully.
            Console.WriteLine( "{0,31}", decimalNum );
        }
        catch( Exception ex )
        {
            // Display the exception type if an exception was thrown.
            Console.WriteLine( "{0,31}", GetExceptionType( ex ) );
        }
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the decimal( double ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-34}{1,31}", "Constructor",
            "Value or Exception" );
        Console.WriteLine( "{0,-34}{1,31}", "-----------",
            "------------------" );

        // Construct decimal objects from double values.
        CreateDecimal( 1.23456789E+5, "1.23456789E+5" );
        CreateDecimal( 1.234567890123E+15, "1.234567890123E+15" );
        CreateDecimal( 1.2345678901234567E+25,
            "1.2345678901234567E+25" );
        CreateDecimal( 1.2345678901234567E+35,
            "1.2345678901234567E+35" );
        CreateDecimal( 1.23456789E-5, "1.23456789E-5" );
        CreateDecimal( 1.234567890123E-15, "1.234567890123E-15" );
        CreateDecimal( 1.2345678901234567E-25,
            "1.2345678901234567E-25" );
        CreateDecimal( 1.2345678901234567E-35,
            "1.2345678901234567E-35" );
        CreateDecimal( 1.0 / 7.0, "1.0 / 7.0" );
    }
}

/*
This example of the decimal( double ) constructor
generates the following output.

Constructor                                   Value or Exception
-----------                                   ------------------
decimal( 1.23456789E+5 )                              123456.789
decimal( 1.234567890123E+15 )                   1234567890123000
decimal( 1.2345678901234567E+25 )      1234567890123460000000000
decimal( 1.2345678901234567E+35 )               OverflowException
decimal( 1.23456789E-5 )                          0.0000123456789
decimal( 1.234567890123E-15 )       0.000000000000001234567890123
decimal( 1.2345678901234567E-25 )  0.0000000000000000000000001235
```

```
decimal( 1.2345678901234567E-35 )                                0
decimal( 1.0 / 7.0 )                            0.142857142857143
*/
```

Remarks

This constructor rounds `value` to 15 significant digits using rounding to nearest. This is done even if the number has more than 15 digits and the less significant digits are zero.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(Int32)

Initializes a new instance of Decimal to the value of the specified 32-bit signed integer.

C#

```
public Decimal (int value);
```

Parameters

`value` Int32

The value to represent as a Decimal.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with an Int32 value.

C#

```
// Example of the decimal( int ) constructor.
using System;

class DecimalCtorIDemo
{
    // Create a decimal object and display its value.
    public static void CreateDecimal( int value, string valToStr )
    {
        decimal decimalNum = new decimal( value );

        // Format the constructor for display.
        string ctor = String.Format( "decimal( {0} )", valToStr );

        // Display the constructor and its value.
        Console.WriteLine( "{0,-30}{1,16}", ctor, decimalNum );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the decimal( int ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-30}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-30}{1,16}", "-----------", "-----" );

        // Construct decimal objects from int values.
        CreateDecimal( int.MinValue, "int.MinValue" );
        CreateDecimal( int.MaxValue, "int.MaxValue" );
        CreateDecimal( 0, "0" );
        CreateDecimal( 999999999, "999999999" );
        CreateDecimal( 0x40000000, "0x40000000" );
        CreateDecimal( unchecked( (int)0xC0000000 ),
            "(int)0xC0000000" );
    }
}

/*
This example of the decimal( int ) constructor
generates the following output.

Constructor                              Value
-----------                              -----
decimal( int.MinValue )            -2147483648
decimal( int.MaxValue )             2147483647
decimal( 0 )                                 0
decimal( 999999999 )                 999999999
decimal( 0x40000000 )               1073741824
```

```
decimal( (int)0xC0000000 )          -1073741824
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(Int32[])

Initializes a new instance of Decimal to a decimal value represented in binary and contained in a specified array.

C#

```
public Decimal (int[] bits);
```

Parameters

`bits` Int32[]

An array of 32-bit signed integers containing a representation of a decimal value.

Exceptions

ArgumentNullException

`bits` is `null`.

ArgumentException

The length of the `bits` is not 4.

-or-

The representation of the decimal value in `bits` is not valid.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with an array of four Int32 values.

C#

```
// Example of the decimal( int[ ] ) constructor.
using System;

class DecimalCtorIArrDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Create a decimal object and display its value.
    public static void CreateDecimal( int[ ] bits )
    {
        // Format the constructor for display.
        string ctor = String.Format(
            "decimal( {{ 0x{0:X}", bits[ 0 ] );
        string valOrExc;

        for( int index = 1; index < bits.Length; index++ )
            ctor += String.Format( ", 0x{0:X}", bits[ index ] );
        ctor += " } )";

        try
        {
            // Construct the decimal value.
            decimal decimalNum = new decimal( bits );

            // Format the decimal value for display.
            valOrExc = decimalNum.ToString( );
        }
        catch( Exception ex )
        {
            // Save the exception type if an exception was thrown.
            valOrExc = GetExceptionType( ex );
        }

        // Display the constructor and decimal value or exception.
        int ctorLen = 76 - valOrExc.Length;

        // Display the data on one line if it will fit.
```

```
        if( ctorLen > ctor.Length )
            Console.WriteLine( "{0}{1}", ctor.PadRight( ctorLen ),
                valOrExc );

        // Otherwise, display the data on two lines.
        else
        {
            Console.WriteLine( "{0}", ctor );
            Console.WriteLine( "{0,76}", valOrExc );
        }
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the decimal( int[ ] ) constructor " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( "{0,-38}{1,38}", "Constructor",
            "Value or Exception" );
        Console.WriteLine( "{0,-38}{1,38}", "-----------",
            "------------------" );

        // Construct decimal objects from integer arrays.
        CreateDecimal( new int[ ] { 0, 0, 0, 0 } );
        CreateDecimal( new int[ ] { 0, 0, 0 } );
        CreateDecimal( new int[ ] { 0, 0, 0, 0, 0 } );
        CreateDecimal( new int[ ] { 1000000000, 0, 0, 0 } );
        CreateDecimal( new int[ ] { 0, 1000000000, 0, 0 } );
        CreateDecimal( new int[ ] { 0, 0, 1000000000, 0 } );
        CreateDecimal( new int[ ] { 0, 0, 0, 1000000000 } );
        CreateDecimal( new int[ ] { -1, -1, -1, 0 } );
        CreateDecimal( new int[ ]
            { -1, -1, -1, unchecked( (int)0x80000000 ) } );
        CreateDecimal( new int[ ] { -1, 0, 0, 0x100000 } );
        CreateDecimal( new int[ ] { -1, 0, 0, 0x1C0000 } );
        CreateDecimal( new int[ ] { -1, 0, 0, 0x1D0000 } );
        CreateDecimal( new int[ ] { -1, 0, 0, 0x1C0001 } );
        CreateDecimal( new int[ ]
            { 0xF0000, 0xF0000, 0xF0000, 0xF0000 } );
    }
}

/*
This example of the decimal( int[ ] ) constructor
generates the following output.

Constructor                                            Value or
Exception
-----------                                            ---------------
---
decimal( { 0x0, 0x0, 0x0, 0x0 } )
0
decimal( { 0x0, 0x0, 0x0 } )
ArgumentException
decimal( { 0x0, 0x0, 0x0, 0x0, 0x0 } )
```

```
ArgumentException
decimal( { 0x3B9ACA00, 0x0, 0x0, 0x0 } )
1000000000
decimal( { 0x0, 0x3B9ACA00, 0x0, 0x0 } )
4294967296000000000
decimal( { 0x0, 0x0, 0x3B9ACA00, 0x0 } )
18446744073709551616000000000
decimal( { 0x0, 0x0, 0x0, 0x3B9ACA00 } )
ArgumentException
decimal( { 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x0 } )

79228162514264337593543950335
decimal( { 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0x80000000 } )

-79228162514264337593543950335
decimal( { 0xFFFFFFFF, 0x0, 0x0, 0x100000 } )
0.0000004294967295
decimal( { 0xFFFFFFFF, 0x0, 0x0, 0x1C0000 } )
0.0000000000000000004294967295
decimal( { 0xFFFFFFFF, 0x0, 0x0, 0x1D0000 } )
ArgumentException
decimal( { 0xFFFFFFFF, 0x0, 0x0, 0x1C0001 } )
ArgumentException
decimal( { 0xF0000, 0xF0000, 0xF0000, 0xF0000 } )

18133887298.441562272235520
*/
```

Remarks

The binary representation of a Decimal number consists of a 1-bit sign, a 96-bit integer number, and a scaling factor used to divide the integer number and specify what portion of it is a decimal fraction. The scaling factor is implicitly the number 10, raised to an exponent ranging from 0 to 28.

`bits` is a four-element long array of 32-bit signed integers.

`bits` [0], `bits` [1], and `bits` [2] contain the low, middle, and high 32 bits of the 96-bit integer number.

`bits` [3] contains the scale factor and sign, and consists of following parts:

Bits 0 to 15, the lower word, are unused and must be zero.

Bits 16 to 23 must contain an exponent between 0 and 28, which indicates the power of 10 to divide the integer number.

Bits 24 to 30 are unused and must be zero.

Bit 31 contains the sign; 0 meaning positive, and 1 meaning negative.

A numeric value might have several possible binary representations; all are equally valid and numerically equivalent. Note that the bit representation differentiates between negative and positive zero. These values are treated as being equal in all operations.

See also

- GetBits(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(Int64)

Initializes a new instance of Decimal to the value of the specified 64-bit signed integer.

C#

```
public Decimal (long value);
```

Parameters

`value` Int64

The value to represent as a Decimal.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with an Int64 value.

C#

```
// Example of the decimal( long ) constructor.
using System;

class DecimalCtorLDemo
{
    // Create a decimal object and display its value.
    public static void CreateDecimal( long value, string valToStr )
    {
        decimal decimalNum = new decimal( value );

        // Format the constructor for display.
        string ctor = String.Format( "decimal( {0} )", valToStr );

        // Display the constructor and its value.
        Console.WriteLine( "{0,-35}{1,22}", ctor, decimalNum );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the decimal( long ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-35}{1,22}", "Constructor", "Value" );
        Console.WriteLine( "{0,-35}{1,22}", "-----------", "-----" );

        // Construct decimal objects from long values.
        CreateDecimal( long.MinValue, "long.MinValue" );
        CreateDecimal( long.MaxValue, "long.MaxValue" );
        CreateDecimal( 0L, "0L" );
        CreateDecimal( 999999999999999999, "999999999999999999" );
        CreateDecimal( 0x2000000000000000, "0x2000000000000000" );
        CreateDecimal( unchecked( (long)0xE000000000000000 ),
            "(long)0xE000000000000000" );
    }
}

/*
This example of the decimal( long ) constructor
generates the following output.

Constructor                                          Value
-----------                                          -----
decimal( long.MinValue )             -9223372036854775808
decimal( long.MaxValue )              9223372036854775807
decimal( 0 )                                            0
decimal( 999999999999999999 )           999999999999999999
decimal( 0x2000000000000000 )         2305843009213693952
```

```
decimal( (long)0xE000000000000000 )  -2305843009213693952
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(ReadOnlySpan<Int32>)

Initializes a new instance of Decimal to a decimal value represented in binary and contained in the specified span.

C#

```
public Decimal (ReadOnlySpan<int> bits);
```

Parameters

`bits` ReadOnlySpan<Int32>

A span of four Int32 values that contains a binary representation of a decimal value.

Exceptions

ArgumentException

The length of `bits` is not 4, or the representation of the decimal value in `bits` is not valid.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Decimal(Single)

Initializes a new instance of Decimal to the value of the specified single-precision floating-point number.

```
C#

public Decimal (float value);
```

Parameters

`value` Single

The value to represent as a Decimal.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

-or-

`value` is NaN, PositiveInfinity, or NegativeInfinity.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with a Single value.

```
C#

// Example of the decimal( float ) constructor.
using System;

class DecimalCtorSDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
```

```
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Create a decimal object and display its value.
    public static void CreateDecimal( float value, string valToStr )
    {
        // Format and display the constructor.
        Console.Write( "{0,-27}",
            String.Format( "decimal( {0} )", valToStr ) );

        try
        {
            // Construct the decimal value.
            decimal decimalNum = new decimal( value );

            // Display the value if it was created successfully.
            Console.WriteLine( "{0,31}", decimalNum );
        }
        catch( Exception ex )
        {
            // Display the exception type if an exception was thrown.
            Console.WriteLine( "{0,31}", GetExceptionType( ex ) );
        }
    }

    public static void Main( )
    {

        Console.WriteLine( "This example of the decimal( float ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-27}{1,31}", "Constructor",
            "Value or Exception" );
        Console.WriteLine( "{0,-27}{1,31}", "-----------",
            "------------------" );

        // Construct decimal objects from float values.
        CreateDecimal( 1.2345E+5F, "1.2345E+5F" );
        CreateDecimal( 1.234567E+15F, "1.234567E+15F" );
        CreateDecimal( 1.23456789E+25F, "1.23456789E+25F" );
        CreateDecimal( 1.23456789E+35F, "1.23456789E+35F" );
        CreateDecimal( 1.2345E-5F, "1.2345E-5F" );
        CreateDecimal( 1.234567E-15F, "1.234567E-15F" );
        CreateDecimal( 1.23456789E-25F, "1.23456789E-25F" );
        CreateDecimal( 1.23456789E-35F, "1.23456789E-35F" );
        CreateDecimal( 1.0F / 7.0F, "1.0F / 7.0F" );
    }
}

/*
This example of the decimal( float ) constructor
generates the following output.

Constructor                                Value or Exception
-----------                                ------------------
decimal( 1.2345E+5F )                                  123450
```

```
decimal( 1.234567E+15F )                  1234567000000000
decimal( 1.23456789E+25F )      12345680000000000000000000
decimal( 1.23456789E+35F )               OverflowException
decimal( 1.2345E-5F )                          0.000012345
decimal( 1.234567E-15F )           0.00000000000001234567
decimal( 1.23456789E-25F )  0.0000000000000000000000001235
decimal( 1.23456789E-35F )                               0
decimal( 1.0F / 7.0F )                           0.1428571
*/
```

Remarks

This constructor rounds `value` to 7 significant digits using rounding to nearest. This is done even if the number has more than 7 digits and the less significant digits are zero.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(UInt32)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of Decimal to the value of the specified 32-bit unsigned integer.

```
C#
```

```
[System.CLSCompliant(false)]
public Decimal (uint value);
```

Parameters

value UInt32

The value to represent as a Decimal.

Attributes CLSCompliantAttribute

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with a UInt32 value.

C#

```
// Example of the decimal( uint ) constructor.
using System;

class DecimalCtorUIDemo
{
    // Create a decimal object and display its value.
    public static void CreateDecimal( uint value, string valToStr )
    {
        decimal decimalNum = new decimal( value );

        // Format the constructor for display.
        string ctor = String.Format( "decimal( {0} )", valToStr );

        // Display the constructor and its value.
        Console.WriteLine( "{0,-33}{1,16}", ctor, decimalNum );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the decimal( uint ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-33}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-33}{1,16}", "-----------", "-----" );

        // Construct decimal objects from uint values.
        CreateDecimal( uint.MinValue, "uint.MinValue" );
        CreateDecimal( uint.MaxValue, "uint.MaxValue" );
        CreateDecimal( (uint)int.MaxValue, "(uint)int.MaxValue" );
        CreateDecimal( 999999999U, "999999999U" );
        CreateDecimal( 0x40000000U, "0x40000000U" );
        CreateDecimal( 0xC0000000, "0xC0000000" );
```

```
    }
}

/*
This example of the decimal( uint ) constructor
generates the following output.

Constructor                                 Value
-----------                                 -----
decimal( uint.MinValue )                        0
decimal( uint.MaxValue )               4294967295
decimal( (uint)int.MaxValue )          2147483647
decimal( 999999999U )                   999999999
decimal( 0x40000000U )                 1073741824
decimal( 0xC0000000 )                  3221225472
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(UInt64)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of Decimal to the value of the specified 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
```

```
public Decimal (ulong value);
```

Parameters

value UInt64

The value to represent as a Decimal.

Attributes CLSCompliantAttribute

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with a UInt64 value.

C#

```
// Example of the decimal( ulong ) constructor.
using System;

class DecimalCtorLDemo
{
    // Create a decimal object and display its value.
    public static void CreateDecimal( ulong value, string valToStr )
    {
        decimal decimalNum = new decimal( value );

        // Format the constructor for display.
        string ctor = String.Format( "decimal( {0} )", valToStr );

        // Display the constructor and its value.
        Console.WriteLine( "{0,-35}{1,22}", ctor, decimalNum );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the decimal( ulong ) " +
            "constructor \ngenerates the following output.\n" );
        Console.WriteLine( "{0,-35}{1,22}", "Constructor", "Value" );
        Console.WriteLine( "{0,-35}{1,22}", "-----------", "-----" );

        // Construct decimal objects from ulong values.
        CreateDecimal( ulong.MinValue, "ulong.MinValue" );
        CreateDecimal( ulong.MaxValue, "ulong.MaxValue" );
        CreateDecimal( long.MaxValue, "long.MaxValue" );
        CreateDecimal( 999999999999999999, "999999999999999999" );
        CreateDecimal( 0x2000000000000000, "0x2000000000000000" );
        CreateDecimal( 0xE000000000000000, "0xE000000000000000" );
    }
}
```

```
/*
This example of the decimal( ulong ) constructor
generates the following output.

Constructor                                          Value
-----------                                          -----
decimal( ulong.MinValue )                                0
decimal( ulong.MaxValue )            18446744073709551615
decimal( long.MaxValue )              9223372036854775807
decimal( 999999999999999999 )          999999999999999999
decimal( 0x2000000000000000 )         2305843009213693952
decimal( 0xE000000000000000 )        16140901064495857664
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Decimal(Int32, Int32, Int32, Boolean, Byte)

Initializes a new instance of Decimal from parameters specifying the instance's constituent parts.

C#

```
public Decimal (int lo, int mid, int hi, bool isNegative, byte scale);
```

Parameters

lo Int32

The low 32 bits of a 96-bit integer.

mid Int32

The middle 32 bits of a 96-bit integer.

hi Int32

The high 32 bits of a 96-bit integer.

isNegative Boolean

`true` to indicate a negative number; `false` to indicate a positive number.

scale Byte

A power of 10 ranging from 0 to 28.

Exceptions

ArgumentOutOfRangeException

`scale` is greater than 28.

Examples

The following code example creates several `Decimal` numbers using the constructor overload that initializes a `Decimal` structure with three Int32 value words, a Boolean sign, and a Byte scale factor.

C#

```
// Example of the decimal( int, int, int, bool, byte ) constructor.
using System;

class DecimalCtorIIIBByDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Create a decimal object and display its value.
    public static void CreateDecimal( int low, int mid, int high,
        bool isNeg, byte scale )
    {
        // Format the constructor for display.
        string ctor = String.Format(
            "decimal( {0}, {1}, {2}, {3}, {4} )",
            low, mid, high, isNeg, scale );
```

```
        string valOrExc;

        try
        {
            // Construct the decimal value.
            decimal decimalNum = new decimal(
                low, mid, high, isNeg, scale );

            // Format and save the decimal value.
            valOrExc = decimalNum.ToString( );
        }
        catch( Exception ex )
        {
            // Save the exception type if an exception was thrown.
            valOrExc = GetExceptionType( ex );
        }

        // Display the constructor and decimal value or exception.
        int ctorLen = 76 - valOrExc.Length;

        // Display the data on one line if it will fit.
        if ( ctorLen > ctor.Length )
            Console.WriteLine( "{0}{1}", ctor.PadRight( ctorLen ),
                valOrExc );

        // Otherwise, display the data on two lines.
        else
        {
            Console.WriteLine( "{0}", ctor );
            Console.WriteLine( "{0,76}", valOrExc );
        }
    }

    public static void Main( )
    {

        Console.WriteLine( "This example of the decimal( int, int, " +
            "int, bool, byte ) \nconstructor " +
            "generates the following output.\n" );
        Console.WriteLine( "{0,-38}{1,38}", "Constructor",
            "Value or Exception" );
        Console.WriteLine( "{0,-38}{1,38}", "-----------",
            "-----------------" );

        // Construct decimal objects from the component fields.
        CreateDecimal( 0, 0, 0, false, 0 );
        CreateDecimal( 0, 0, 0, false, 27 );
        CreateDecimal( 0, 0, 0, true, 0 );
        CreateDecimal( 1000000000, 0, 0, false, 0 );
        CreateDecimal( 0, 1000000000, 0, false, 0 );
        CreateDecimal( 0, 0, 1000000000, false, 0 );
        CreateDecimal( 1000000000, 1000000000, 1000000000, false, 0 );
        CreateDecimal( -1, -1, -1, false, 0 );
        CreateDecimal( -1, -1, -1, true, 0 );
        CreateDecimal( -1, -1, -1, false, 15 );
```

```
        CreateDecimal( -1, -1, -1, false, 28 );
        CreateDecimal( -1, -1, -1, false, 29 );
        CreateDecimal( int.MaxValue, 0, 0, false, 18 );
        CreateDecimal( int.MaxValue, 0, 0, false, 28 );
        CreateDecimal( int.MaxValue, 0, 0, true, 28 );
    }
}

/*
This example of the decimal( int, int, int, bool, byte )
constructor generates the following output.

Constructor                                                    Value or
Exception
-----------                                                    ---------------
---
decimal( 0, 0, 0, False, 0 )
0
decimal( 0, 0, 0, False, 27 )
0
decimal( 0, 0, 0, True, 0 )
0
decimal( 1000000000, 0, 0, False, 0 )
1000000000
decimal( 0, 1000000000, 0, False, 0 )
4294967296000000000
decimal( 0, 0, 1000000000, False, 0 )
18446744073709551616000000000
decimal( 1000000000, 1000000000, 1000000000, False, 0 )

18446744078004518913000000000
decimal( -1, -1, -1, False, 0 )
79228162514264337593543950335
decimal( -1, -1, -1, True, 0 )
-79228162514264337593543950335
decimal( -1, -1, -1, False, 15 )
79228162514264.337593543950335
decimal( -1, -1, -1, False, 28 )
7.9228162514264337593543950335
decimal( -1, -1, -1, False, 29 )
ArgumentOutOfRangeException
decimal( 2147483647, 0, 0, False, 18 )
0.000000002147483647
decimal( 2147483647, 0, 0, False, 28 )
0.0000000000000000002147483647
decimal( 2147483647, 0, 0, True, 28 )
-0.0000000000000000002147483647
*/
```

The following example uses the GetBits method to retrieve the component parts of an array. It then uses this array in the call to the Decimal(Int32, Int32, Int32, Boolean, Byte) constructor to instantiate a new Decimal value.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal[] values = { 1234.96m, -1234.96m };
      foreach (var value in values) {
         int[] parts = Decimal.GetBits(value);
         bool sign = (parts[3] & 0x80000000) != 0;

         byte scale = (byte) ((parts[3] >> 16) & 0x7F);
         Decimal newValue = new Decimal(parts[0], parts[1], parts[2],
sign, scale);
         Console.WriteLine("{0} --> {1}", value, newValue);
      }
   }
}
// The example displays the following output:
//       1234.96 --> 1234.96
//       -1234.96 --> -1234.96
```

Remarks

The binary representation of a Decimal number consists of a 1-bit sign, a 96-bit integer number, and a scaling factor used to divide the integer number and specify what portion of it is a decimal fraction. The scaling factor is implicitly the number 10 raised to an exponent ranging from 0 to 28.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of Decimal. This field is constant and read-only.

C#

```
public static readonly decimal MaxValue;
```

Field Value

Decimal

Examples

The following code sample illustrates the use of `MaxValue` :

C#

```
class PiggyBank {
    public decimal Capacity {
        get {
            return Decimal.MaxValue;
        }
    }

    protected decimal MyFortune;

    public void AddPenny() {
        MyFortune += .01m;
    }
}
```

Remarks

The value of this constant is positive 79,228,162,514,264,337,593,543,950,335.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Zero
- One
- MinusOne
- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.MinusOne Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number negative one (-1).

C#

```
public static readonly decimal MinusOne;
```

Field Value

Decimal

Examples

The following code example illustrates the use of the `MinusOne` field.

C#

```
// Example of the Decimal fields.
using System;

class DecimalFieldsDemo
{
    public static void Main( )
    {
        const string numberFmt = "{0,-25}{1,45:N0}";
        const string exprFmt = "{0,-55}{1,15}";

        Console.WriteLine(
            "This example of the fields of the Decimal structure " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field or Expression", "Value" );
        Console.WriteLine( numberFmt, "-------------------", "-----" );

        // Display the values of the Decimal fields.
        Console.WriteLine( numberFmt, "Decimal.MaxValue",
            Decimal.MaxValue );
        Console.WriteLine( numberFmt, "Decimal.MinValue",
            Decimal.MinValue );
```

```
        Console.WriteLine( numberFmt, "Decimal.MinusOne",
            Decimal.MinusOne );
        Console.WriteLine( numberFmt, "Decimal.One", Decimal.One );
        Console.WriteLine( numberFmt, "Decimal.Zero", Decimal.Zero );
        Console.WriteLine( );

        // Display the values of expressions of the Decimal fields.
        Console.WriteLine( exprFmt,
            "( Decimal.MinusOne + Decimal.One ) == Decimal.Zero",
            (Decimal.MinusOne + Decimal.One ) == Decimal.Zero );
        Console.WriteLine( exprFmt,
            "Decimal.MaxValue + Decimal.MinValue",
            Decimal.MaxValue + Decimal.MinValue );
        Console.WriteLine( exprFmt,
            "Decimal.MinValue / Decimal.MaxValue",
            Decimal.MinValue / Decimal.MaxValue );
        Console.WriteLine( "{0,-40}{1,30}",
            "100000000000000M / Decimal.MaxValue",
            100000000000000M / Decimal.MaxValue );
    }
}

/*
This example of the fields of the Decimal structure
generates the following output.

Field or Expression                                          Value
-------------------                                          -----
Decimal.MaxValue                 79,228,162,514,264,337,593,543,950,335
Decimal.MinValue                -79,228,162,514,264,337,593,543,950,335
Decimal.MinusOne                                                    -1
Decimal.One                                                          1
Decimal.Zero                                                         0

( Decimal.MinusOne + Decimal.One ) == Decimal.Zero               True
Decimal.MaxValue + Decimal.MinValue                                  0
Decimal.MinValue / Decimal.MaxValue                                 -1
100000000000000M / Decimal.MaxValue     0.0000000000000012621774483536
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Zero
- One
- MaxValue
- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of Decimal. This field is constant and read-only.

C#

```
public static readonly decimal MinValue;
```

Field Value

Decimal

Examples

The following code example illustrates the use of the `MinValue` field.

C#

```
// Example of the Decimal fields.
using System;

class DecimalFieldsDemo
{
    public static void Main( )
    {
        const string numberFmt = "{0,-25}{1,45:N0}";
        const string exprFmt = "{0,-55}{1,15}";

        Console.WriteLine(
            "This example of the fields of the Decimal structure " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field or Expression", "Value" );
        Console.WriteLine( numberFmt, "-------------------", "-----" );

        // Display the values of the Decimal fields.
        Console.WriteLine( numberFmt, "Decimal.MaxValue",
            Decimal.MaxValue );
        Console.WriteLine( numberFmt, "Decimal.MinValue",
            Decimal.MinValue );
```

```
        Console.WriteLine( numberFmt, "Decimal.MinusOne",
            Decimal.MinusOne );
        Console.WriteLine( numberFmt, "Decimal.One", Decimal.One );
        Console.WriteLine( numberFmt, "Decimal.Zero", Decimal.Zero );
        Console.WriteLine( );

        // Display the values of expressions of the Decimal fields.
        Console.WriteLine( exprFmt,
            "( Decimal.MinusOne + Decimal.One ) == Decimal.Zero",
            (Decimal.MinusOne + Decimal.One ) == Decimal.Zero );
        Console.WriteLine( exprFmt,
            "Decimal.MaxValue + Decimal.MinValue",
            Decimal.MaxValue + Decimal.MinValue );
        Console.WriteLine( exprFmt,
            "Decimal.MinValue / Decimal.MaxValue",
            Decimal.MinValue / Decimal.MaxValue );
        Console.WriteLine( "{0,-40}{1,30}",
            "100000000000000M / Decimal.MaxValue",
            100000000000000M / Decimal.MaxValue );
    }
}

/*
This example of the fields of the Decimal structure
generates the following output.

Field or Expression                                          Value
-------------------                                          -----
Decimal.MaxValue                 79,228,162,514,264,337,593,543,950,335
Decimal.MinValue                -79,228,162,514,264,337,593,543,950,335
Decimal.MinusOne                                                    -1
Decimal.One                                                          1
Decimal.Zero                                                         0

( Decimal.MinusOne + Decimal.One ) == Decimal.Zero               True
Decimal.MaxValue + Decimal.MinValue                                  0
Decimal.MinValue / Decimal.MaxValue                                 -1
100000000000000M / Decimal.MaxValue     0.0000000000000012621774483536
*/
```

Remarks

The value of this constant is negative 79,228,162,514,264,337,593,543,950,335.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Zero
- One
- MinusOne
- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.One Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number one (1).

C#

```
public static readonly decimal One;
```

Field Value

Decimal

Examples

The following code example illustrates the use of the `One` field.

C#

```
// Example of the Decimal fields.
using System;

class DecimalFieldsDemo
{
    public static void Main( )
    {
        const string numberFmt = "{0,-25}{1,45:N0}";
        const string exprFmt = "{0,-55}{1,15}";

        Console.WriteLine(
            "This example of the fields of the Decimal structure " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field or Expression", "Value" );
        Console.WriteLine( numberFmt, "-------------------", "-----" );

        // Display the values of the Decimal fields.
        Console.WriteLine( numberFmt, "Decimal.MaxValue",
            Decimal.MaxValue );
        Console.WriteLine( numberFmt, "Decimal.MinValue",
            Decimal.MinValue );
```

```
        Console.WriteLine( numberFmt, "Decimal.MinusOne",
            Decimal.MinusOne );
        Console.WriteLine( numberFmt, "Decimal.One", Decimal.One );
        Console.WriteLine( numberFmt, "Decimal.Zero", Decimal.Zero );
        Console.WriteLine( );

        // Display the values of expressions of the Decimal fields.
        Console.WriteLine( exprFmt,
            "( Decimal.MinusOne + Decimal.One ) == Decimal.Zero",
            (Decimal.MinusOne + Decimal.One ) == Decimal.Zero );
        Console.WriteLine( exprFmt,
            "Decimal.MaxValue + Decimal.MinValue",
            Decimal.MaxValue + Decimal.MinValue );
        Console.WriteLine( exprFmt,
            "Decimal.MinValue / Decimal.MaxValue",
            Decimal.MinValue / Decimal.MaxValue );
        Console.WriteLine( "{0,-40}{1,30}",
            "100000000000000M / Decimal.MaxValue",
            100000000000000M / Decimal.MaxValue );
    }
}

/*
This example of the fields of the Decimal structure
generates the following output.

Field or Expression                                          Value
-------------------                                          -----
Decimal.MaxValue                 79,228,162,514,264,337,593,543,950,335
Decimal.MinValue                -79,228,162,514,264,337,593,543,950,335
Decimal.MinusOne                                                    -1
Decimal.One                                                          1
Decimal.Zero                                                         0

( Decimal.MinusOne + Decimal.One ) == Decimal.Zero               True
Decimal.MaxValue + Decimal.MinValue                                  0
Decimal.MinValue / Decimal.MaxValue                                 -1
100000000000000M / Decimal.MaxValue     0.0000000000000012621774483536
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Zero
- MinusOne
- MaxValue
- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Zero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number zero (0).

C#

```
public static readonly decimal Zero;
```

Field Value

Decimal

Examples

The following code example illustrates the use of the `Zero` field.

C#

```
// Example of the Decimal fields.
using System;

class DecimalFieldsDemo
{
    public static void Main( )
    {
        const string numberFmt = "{0,-25}{1,45:N0}";
        const string exprFmt = "{0,-55}{1,15}";

        Console.WriteLine(
            "This example of the fields of the Decimal structure " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field or Expression", "Value" );
        Console.WriteLine( numberFmt, "-------------------", "-----" );

        // Display the values of the Decimal fields.
        Console.WriteLine( numberFmt, "Decimal.MaxValue",
            Decimal.MaxValue );
        Console.WriteLine( numberFmt, "Decimal.MinValue",
            Decimal.MinValue );
```

```
        Console.WriteLine( numberFmt, "Decimal.MinusOne",
            Decimal.MinusOne );
        Console.WriteLine( numberFmt, "Decimal.One", Decimal.One );
        Console.WriteLine( numberFmt, "Decimal.Zero", Decimal.Zero );
        Console.WriteLine( );

        // Display the values of expressions of the Decimal fields.
        Console.WriteLine( exprFmt,
            "( Decimal.MinusOne + Decimal.One ) == Decimal.Zero",
            (Decimal.MinusOne + Decimal.One ) == Decimal.Zero );
        Console.WriteLine( exprFmt,
            "Decimal.MaxValue + Decimal.MinValue",
            Decimal.MaxValue + Decimal.MinValue );
        Console.WriteLine( exprFmt,
            "Decimal.MinValue / Decimal.MaxValue",
            Decimal.MinValue / Decimal.MaxValue );
        Console.WriteLine( "{0,-40}{1,30}",
            "100000000000000M / Decimal.MaxValue",
            100000000000000M / Decimal.MaxValue );
    }
}

/*
This example of the fields of the Decimal structure
generates the following output.

Field or Expression                                          Value
-------------------                                          -----
Decimal.MaxValue                 79,228,162,514,264,337,593,543,950,335
Decimal.MinValue                -79,228,162,514,264,337,593,543,950,335
Decimal.MinusOne                                                    -1
Decimal.One                                                          1
Decimal.Zero                                                         0

( Decimal.MinusOne + Decimal.One ) == Decimal.Zero               True
Decimal.MaxValue + Decimal.MinValue                                  0
Decimal.MinValue / Decimal.MaxValue                                 -1
100000000000000M / Decimal.MaxValue     0.0000000000000012621774483536
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- One
- MinusOne
- MaxValue
- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Scale Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the scaling factor of the decimal, which is a number from 0 to 28 that represents the number of decimal digits.

C#

```
public byte Scale { get; }
```

Property Value

Byte

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Abs(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static decimal Abs (decimal value);
```

Parameters

`value` Decimal

The value for which to get its absolute.

Returns

Decimal

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Add(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two specified Decimal values.

C#

```
public static decimal Add (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to add.

d2 Decimal

The second value to add.

Returns

Decimal

The sum of `d1` and `d2`.

Exceptions

OverflowException

The sum of `d1` and `d2` is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The following code sample illustrates the use of `Add` :

C#

```
class PiggyBank {
    public void AddPenny() {
        MyFortune = Decimal.Add(MyFortune, .01m);
    }

    public override string ToString() {
        return MyFortune.ToString("C")+" in piggy bank";
    }

    protected decimal MyFortune;
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(Decimal, Decimal)
- Multiply(Decimal, Decimal)
- Divide(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Ceiling(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smallest integral value that is greater than or equal to the specified decimal number.

C#

```
public static decimal Ceiling (decimal d);
```

Parameters

d Decimal

A decimal number.

Returns

Decimal

The smallest integral value that is greater than or equal to the `d` parameter. Note that this method returns a Decimal instead of an integral type.

Implements

Ceiling(TSelf)

Examples

The following example illustrates the Ceiling method and contrasts it with the Floor method.

C#

```
using System;

public class Example
```

```
{
   public static void Main()
   {
      decimal[] values = {12.6m, 12.1m, 9.5m, 8.16m, .1m, -.1m,  -1.1m,
                          -1.9m, -3.9m};
      Console.WriteLine("{0,-8} {1,10} {2,10}\n",
                        "Value", "Ceiling", "Floor");
      foreach (decimal value in values)
      Console.WriteLine("{0,-8} {1,10} {2,10}", value,
                        Decimal.Ceiling(value), Decimal.Floor(value));
   }
}
// The example displays the following output:
//       Value       Ceiling      Floor
//
//       12.6             13         12
//       12.1             13         12
//       9.5              10          9
//       8.16              9          8
//       0.1               1          0
//       -0.1              0         -1
//       -1.1             -1         -2
//       -1.9             -1         -2
//       -3.9             -3         -4
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward positive infinity. In other words, if `d` is positive, the presence of any fractional component causes `d` to be rounded to the next highest integer. If `d` is negative, the rounding operation causes any fractional component of `d` to be discarded. The operation of this method differs from the Floor method, which supports rounding toward negative infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Ceiling(Decimal)
- Floor(Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Clamp(Decimal, Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static decimal Clamp (decimal value, decimal min, decimal max);
```

Parameters

`value` Decimal

The value to clamp.

`min` Decimal

The inclusive minimum to which `value` should clamp.

`max` Decimal

The inclusive maximum to which `value` should clamp.

Returns

Decimal

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Compare(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two specified Decimal values.

C#

```
public static int Compare (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Int32

A signed number indicating the relative values of `d1` and `d2`.

Expand table

Return value	Meaning
Less than zero	`d1` is less than `d2`.
Zero	`d1` and `d2` are equal.
Greater than zero	`d1` is greater than `d2`.

Examples

The following example compares several Decimal values. Note that the first comparison indicates that the two values are equal despite the subtraction operation performed on the `value2` variable. This is because the Decimal type has 29 digits of precision, whereas a difference between these two values can be detected only with 30 digits of precision.

C#

```
using System;

public enum Relationship
{  LessThan = -1, Equals = 0, GreaterThan = 1 }

public class Example
{
   public static void Main()
   {
      decimal value1 = Decimal.MaxValue;
      decimal value2 = value1 - .01m;
      Console.WriteLine("{0} {2} {1}", value1, value2,
                        (Relationship) Decimal.Compare(value1, value2));

      value2 = value1 / 12m - .1m;
      value1 = value1 / 12m;
      Console.WriteLine("{0} {2} {1}", value1, value2,
                        (Relationship) Decimal.Compare(value1, value2));

      value1 = value1 - .2m;
      value2 = value2 + .1m;
      Console.WriteLine("{0} {2} {1}", value1, value2,
                        (Relationship) Decimal.Compare(value1, value2));
   }
}
// The example displays the following output:
//     79228162514264337593543950335 Equals 79228162514264337593543950335
//     6602346876188694799461995861.2 GreaterThan
6602346876188694799461995861.1
//     6602346876188694799461995861.0 LessThan
6602346876188694799461995861.2
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(Object)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Decimal and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Decimal)	Compares this instance to a specified Decimal object and returns a comparison of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns a comparison of their relative values.

CompareTo(Decimal)

Compares this instance to a specified Decimal object and returns a comparison of their relative values.

C#

```
public int CompareTo (decimal value);
```

Parameters

`value` Decimal

The object to compare with this instance.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return value	Meaning
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");
```

```
    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
```

```
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                            resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
```

```
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Decimal.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the parameter as an Int32 object, then generates a Int32.CompareTo method to compare the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- Compare(Decimal, Decimal)

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns a comparison of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

The object to compare with this instance, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return value	Meaning
Less than zero	This instance is less than `value`.

Return value	Meaning
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Decimal.

Examples

The following code example compares several `Decimal` and other objects to a reference `Decimal` value using the `CompareTo` method.

C#

```
// Example of the decimal.CompareTo and decimal.Equals instance
// methods.
using System;

class DecCompToEqualsObjDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Compare the decimal to the object parameters,
    // and display the object parameters with the results.
    public static void CompDecimalToObject( decimal Left,
        object Right, string RightText )
    {

        Console.WriteLine( "{0,-46}{1}", "object: "+RightText,
            Right );
        Console.WriteLine( "{0,-46}{1}", "Left.Equals( object )",
            Left.Equals( Right ) );
        Console.Write( "{0,-46}", "Left.CompareTo( object )" );
```

```
        try
        {
            // Catch the exception if CompareTo( ) throws one.
            Console.WriteLine( "{0}\n", Left.CompareTo( Right ) );
        }
        catch( Exception ex )
        {
            Console.WriteLine( "{0}\n", GetExceptionType( ex ) );
        }
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the decimal.Equals( object ) and \n" +
            "decimal.CompareTo( object ) methods generates the \n" +
            "following output. It creates several different " +
            "decimal \nvalues and compares them with the following " +
            "reference value.\n" );

        // Create a reference decimal value.
        decimal Left = new decimal( 987.654 );

        Console.WriteLine( "{0,-46}{1}\n",
            "Left: decimal( 987.654 )", Left );

        // Create objects to compare with the reference.
        CompDecimalToObject( Left, new decimal( 9.8765400E+2 ),
            "decimal( 9.8765400E+2 )" );
        CompDecimalToObject( Left, 987.6541M, "987.6541D" );
        CompDecimalToObject( Left, 987.6539M, "987.6539D" );
        CompDecimalToObject( Left,
            new decimal( 987654000, 0, 0, false, 6 ),
            "decimal( 987654000, 0, 0, false, 6 )" );
        CompDecimalToObject( Left, 9.8765400E+2,
            "Double 9.8765400E+2" );
        CompDecimalToObject( Left, "987.654", "String \"987.654\"" );
    }
}

/*
This example of the decimal.Equals( object ) and
decimal.CompareTo( object ) methods generates the
following output. It creates several different decimal
values and compares them with the following reference value.

Left: decimal( 987.654 )                      987.654

object: decimal( 9.8765400E+2 )               987.654
Left.Equals( object )                         True
Left.CompareTo( object )                      0

object: 987.6541D                             987.6541
Left.Equals( object )                         False
Left.CompareTo( object )                      -1
```

```
object: 987.6539D                              987.6539
Left.Equals( object )                          False
Left.CompareTo( object )                       1

object: decimal( 987654000, 0, 0, false, 6 )   987.654000
Left.Equals( object )                          True
Left.CompareTo( object )                       0

object: Double 9.8765400E+2                    987.654
Left.Equals( object )                          False
Left.CompareTo( object )                       ArgumentException

object: String "987.654"                       987.654
Left.Equals( object )                          False
Left.CompareTo( object )                       ArgumentException
*/
```

Remarks

Any instance of Decimal, regardless of its value, is considered greater than `null`.

Parameter `value` must be `null` or an instance of Decimal; otherwise, an exception is thrown.

See also

- Compare(Decimal, Decimal)
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.CopySign(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
public static decimal CopySign (decimal value, decimal sign);
```

Parameters

`value` Decimal

The value whose magnitude is used in the result.

`sign` Decimal

The value whose sign is used in the result.

Returns

Decimal

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static decimal CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Decimal

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.CreateSaturating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static decimal CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Decimal

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static decimal CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Decimal

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Divide(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two specified Decimal values.

C#

```
public static decimal Divide (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The dividend.

d2 Decimal

The divisor.

Returns

Decimal

The result of dividing `d1` by `d2`.

Exceptions

DivideByZeroException

`d2` is zero.

OverflowException

The return value (that is, the quotient) is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example calls the Divide method to divide a range of values by 22.1.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Divide a series of numbers by 22.1
      Decimal dividend = 1230.0m;
      Decimal divisor = 22.1m;
      for (int ctr = 0; ctr <= 10; ctr++) {
         Console.WriteLine("{0:N1} / {1:N1} = {2:N4}", dividend, divisor,
                           Decimal.Divide(dividend, divisor));
         dividend += .1m;
      }
   }
}
// The example displays the following output:
//       1,230.0 / 22.1 = 55.6561
//       1,230.1 / 22.1 = 55.6606
//       1,230.2 / 22.1 = 55.6652
//       1,230.3 / 22.1 = 55.6697
//       1,230.4 / 22.1 = 55.6742
//       1,230.5 / 22.1 = 55.6787
//       1,230.6 / 22.1 = 55.6833
//       1,230.7 / 22.1 = 55.6878
//       1,230.8 / 22.1 = 55.6923
//       1,230.9 / 22.1 = 55.6968
//       1,231.0 / 22.1 = 55.7014
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(Decimal, Decimal)
- Subtract(Decimal, Decimal)
- Multiply(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two instances of Decimal represent the same value.

Overloads

Expand table

Equals(Decimal, Decimal)	Returns a value indicating whether two specified instances of Decimal represent the same value.
Equals(Object)	Returns a value indicating whether this instance and a specified Object represent the same type and value.
Equals(Decimal)	Returns a value indicating whether this instance and a specified Decimal object represent the same value.

Equals(Decimal, Decimal)

Returns a value indicating whether two specified instances of Decimal represent the same value.

C#

```
public static bool Equals (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` and `d2` are equal; otherwise, `false`.

Examples

The following code example compares several `Decimal` values to a reference `Decimal` value using the static `Equals` method.

C#

```
// Example of the decimal.Compare and static decimal.Equals methods.
using System;

class DecCompareEqualsDemo
{
    const string dataFmt = "{0,-45}{1}";

    // Compare decimal parameters, and display them with the results.
    public static void CompareDecimals( decimal Left, decimal Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: "+RightText, Right );
        Console.WriteLine( dataFmt, "decimal.Equals( Left, Right )",
            Decimal.Equals( Left, Right ) );
        Console.WriteLine( dataFmt, "decimal.Compare( Left, Right )",
            Decimal.Compare( Left, Right ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the " +
            "decimal.Equals( decimal, decimal ) and \n" +
            "decimal.Compare( decimal, decimal ) methods " +
            "generates the \nfollowing output. It creates several " +
            "different decimal \nvalues and compares them with " +
            "the following reference value.\n" );

        // Create a reference decimal value.
        decimal Left = new decimal( 123.456 );

        Console.WriteLine( dataFmt, "Left: decimal( 123.456 )",
            Left );

        // Create decimal values to compare with the reference.
        CompareDecimals( Left, new decimal( 1.2345600E+2 ),
            "decimal( 1.2345600E+2 )" );
        CompareDecimals( Left, 123.4561M, "123.4561M" );
        CompareDecimals( Left, 123.4559M, "123.4559M" );
```

```
        CompareDecimals( Left, 123.456000M, "123.456000M" );
        CompareDecimals( Left,
            new decimal( 123456000, 0, 0, false, 6 ),
            "decimal( 123456000, 0, 0, false, 6 )" );
    }
}

/*
This example of the decimal.Equals( decimal, decimal ) and
decimal.Compare( decimal, decimal ) methods generates the
following output. It creates several different decimal
values and compares them with the following reference value.

Left: decimal( 123.456 )                    123.456

Right: decimal( 1.2345600E+2 )              123.456
decimal.Equals( Left, Right )               True
decimal.Compare( Left, Right )              0

Right: 123.4561M                            123.4561
decimal.Equals( Left, Right )               False
decimal.Compare( Left, Right )              -1

Right: 123.4559M                            123.4559
decimal.Equals( Left, Right )               False
decimal.Compare( Left, Right )              1

Right: 123.456000M                          123.456000
decimal.Equals( Left, Right )               True
decimal.Compare( Left, Right )              0

Right: decimal( 123456000, 0, 0, false, 6 )  123.456000
decimal.Equals( Left, Right )               True
decimal.Compare( Left, Right )              0
*/
```

See also

- Compare(Decimal, Decimal)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance and a specified Object represent the same type and value.

C#

```
public override bool Equals (object? value);
```

Parameters

`value` Object

The object to compare with this instance.

Returns

Boolean

`true` if `value` is a Decimal and equal to this instance; otherwise, `false`.

Examples

The following code example compares several `Decimal` and other objects to a reference `Decimal` value using the `Equals` method.

C#

```
// Example of the decimal.CompareTo and decimal.Equals instance
// methods.
using System;

class DecCompToEqualsObjDemo
{
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
```

```
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Compare the decimal to the object parameters,
    // and display the object parameters with the results.
    public static void CompDecimalToObject( decimal Left,
        object Right, string RightText )
    {

        Console.WriteLine( "{0,-46}{1}", "object: "+RightText,
            Right );
        Console.WriteLine( "{0,-46}{1}", "Left.Equals( object )",
            Left.Equals( Right ) );
        Console.Write( "{0,-46}", "Left.CompareTo( object )" );

        try
        {
            // Catch the exception if CompareTo( ) throws one.
            Console.WriteLine( "{0}\n", Left.CompareTo( Right ) );
        }
        catch( Exception ex )
        {
            Console.WriteLine( "{0}\n", GetExceptionType( ex ) );
        }
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the decimal.Equals( object ) and \n" +
            "decimal.CompareTo( object ) methods generates the \n" +
            "following output. It creates several different " +
            "decimal \nvalues and compares them with the following " +
            "reference value.\n" );

        // Create a reference decimal value.
        decimal Left = new decimal( 987.654 );

        Console.WriteLine( "{0,-46}{1}\n",
            "Left: decimal( 987.654 )", Left );

        // Create objects to compare with the reference.
        CompDecimalToObject( Left, new decimal( 9.8765400E+2 ),
            "decimal( 9.8765400E+2 )" );
        CompDecimalToObject( Left, 987.6541M, "987.6541D" );
        CompDecimalToObject( Left, 987.6539M, "987.6539D" );
        CompDecimalToObject( Left,
            new decimal( 987654000, 0, 0, false, 6 ),
            "decimal( 987654000, 0, 0, false, 6 )" );
        CompDecimalToObject( Left, 9.8765400E+2,
            "Double 9.8765400E+2" );
        CompDecimalToObject( Left, "987.654", "String \"987.654\"" );
```

```
    }
}

/*
This example of the decimal.Equals( object ) and
decimal.CompareTo( object ) methods generates the
following output. It creates several different decimal
values and compares them with the following reference value.

Left: decimal( 987.654 )                      987.654

object: decimal( 9.8765400E+2 )               987.654
Left.Equals( object )                         True
Left.CompareTo( object )                      0

object: 987.6541D                             987.6541
Left.Equals( object )                         False
Left.CompareTo( object )                      -1

object: 987.6539D                             987.6539
Left.Equals( object )                         False
Left.CompareTo( object )                      1

object: decimal( 987654000, 0, 0, false, 6 )  987.654000
Left.Equals( object )                         True
Left.CompareTo( object )                      0

object: Double 9.8765400E+2                   987.654
Left.Equals( object )                         False
Left.CompareTo( object )                      ArgumentException

object: String "987.654"                      987.654
Left.Equals( object )                         False
Left.CompareTo( object )                      ArgumentException
*/
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Decimal is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Decimal) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Decimal value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive integral types, including both signed and unsigned types, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the

Equals(Decimal) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static decimal value = 112m;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,17}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
      TestObjectForEquality(int1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,18}", value.Equals(long1));
      TestObjectForEquality(long1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
      Console.WriteLine("value = ushort1: {0,17}",
value.Equals(ushort1));
      TestObjectForEquality(ushort1);

      uint uint1 = 112;
      Console.WriteLine("value = uint1: {0,19}", value.Equals(uint1));
      TestObjectForEquality(uint1);

      ulong ulong1 = 112;
      Console.WriteLine("value = ulong1: {0,18}", value.Equals(ulong1));
      TestObjectForEquality(ulong1);

      float sng1 = 112;
      Console.WriteLine("value = sng1: {0,21}", value.Equals(sng1));
      TestObjectForEquality(sng1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,21}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }
```

```
    private static void TestObjectForEquality(Object obj)
    {
       Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                         value, value.GetType().Name,
                         obj, obj.GetType().Name,
                         value.Equals(obj));
    }
 }
 // The example displays the following output:
 //       value = byte1:              True
 //       112 (Decimal) = 112 (Byte): False
 //
 //       value = short1:              True
 //       112 (Decimal) = 112 (Int16): False
 //
 //       value = int1:                True
 //       112 (Decimal) = 112 (Int32): False
 //
 //       value = long1:               True
 //       112 (Decimal) = 112 (Int64): False
 //
 //       value = sbyte1:              True
 //       112 (Decimal) = 112 (SByte): False
 //
 //       value = ushort1:              True
 //       112 (Decimal) = 112 (UInt16): False
 //
 //       value = uint1:                True
 //       112 (Decimal) = 112 (UInt32): False
 //
 //       value = ulong1:               True
 //       112 (Decimal) = 112 (UInt64): False
 //
 //       value = sng1:                 False
 //       112 (Decimal) = 112 (Single): False
 //
 //       value = dbl1:                 False
 //       112 (Decimal) = 112 (Double): False
```

See also

- Compare(Decimal, Decimal)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Decimal)

Returns a value indicating whether this instance and a specified Decimal object represent the same value.

```
C#

public bool Equals (decimal value);
```

Parameters

`value` Decimal

An object to compare to this instance.

Returns

Boolean

`true` if `value` is equal to this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `value` parameter to an object.

If `value` has fewer bits (is narrower) than the instance type, some programming languages perform an implicit widening conversion that transforms the value of the parameter into a value with more bits.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method to compare the Int32 instance and parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Decimal is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Decimal) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Decimal value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive integral types, including both signed and unsigned types, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Decimal) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static decimal value = 112m;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,17}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
```

```
        TestObjectForEquality(int1);

        long long1 = 112;
        Console.WriteLine("value = long1: {0,18}", value.Equals(long1));
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,17}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,19}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,18}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        float sng1 = 112;
        Console.WriteLine("value = sng1: {0,21}", value.Equals(sng1));
        TestObjectForEquality(sng1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,21}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:              True
//       112 (Decimal) = 112 (Byte): False
//
//       value = short1:              True
//       112 (Decimal) = 112 (Int16): False
//
//       value = int1:                True
//       112 (Decimal) = 112 (Int32): False
//
//       value = long1:               True
//       112 (Decimal) = 112 (Int64): False
//
//       value = sbyte1:              True
//       112 (Decimal) = 112 (SByte): False
```

```
//
//       value = ushort1:              True
//       112 (Decimal) = 112 (UInt16): False
//
//       value = uint1:                True
//       112 (Decimal) = 112 (UInt32): False
//
//       value = ulong1:               True
//       112 (Decimal) = 112 (UInt64): False
//
//       value = sng1:                 False
//       112 (Decimal) = 112 (Single): False
//
//       value = dbl1:                 False
//       112 (Decimal) = 112 (Double): False
```

See also

- Equals(Object)
- CompareTo
- Compare(Decimal, Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Decimal.Floor(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rounds a specified Decimal number to the closest integer toward negative infinity.

C#

```
public static decimal Floor (decimal d);
```

Parameters

d Decimal

The value to round.

Returns

Decimal

If `d` has a fractional part, the next whole Decimal number toward negative infinity that is less than `d`.

-or-

If `d` doesn't have a fractional part, `d` is returned unchanged. Note that the method returns an integral value of type Decimal.

Implements

Floor(TSelf)

Examples

The following example illustrates the Floor method and contrasts it with the Ceiling method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      decimal[] values = {12.6m, 12.1m, 9.5m, 8.16m, .1m, -.1m,  -1.1m,
                          -1.9m, -3.9m};
      Console.WriteLine("{0,-8} {1,10} {2,10}\n",
                        "Value", "Ceiling", "Floor");
      foreach (decimal value in values)
      Console.WriteLine("{0,-8} {1,10} {2,10}", value,
                        Decimal.Ceiling(value), Decimal.Floor(value));
   }
}
// The example displays the following output:
//       Value       Ceiling      Floor
//
//       12.6             13         12
//       12.1             13         12
//       9.5              10          9
//       8.16              9          8
//       0.1               1          0
//       -0.1              0         -1
//       -1.1             -1         -2
//       -1.9             -1         -2
//       -3.9             -3         -4
```

Remarks

The behavior of this method follows IEEE Standard 754, Section 4. This kind of rounding is sometimes called rounding toward negative infinity. In other words, if `d` is positive, any fractional component is truncated. If `d` is negative, the presence of any fractional component causes it to be rounded to the smaller integer. The operation of this method differs from the Ceiling method, which supports rounding toward positive infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Round(Decimal, Int32)
- Truncate(Decimal)
- Ceiling(Decimal)
- Floor(Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.FromOACurrency(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified 64-bit signed integer, which contains an OLE Automation Currency value, to the equivalent Decimal value.

C#

```
public static decimal FromOACurrency (long cy);
```

Parameters

cy Int64

An OLE Automation Currency value.

Returns

Decimal

A Decimal that contains the equivalent of `cy`.

Examples

The following code example uses the `FromOACurrency` method to convert Int64 fields that contain OLE Automation Currency values to the equivalent `Decimal` numbers.

C#

```
// Example of the decimal.FromOACurrency method.
using System;

class DecimalFromOACurrencyDemo
{
    const string dataFmt = "{0,21}{1,25}";
```

```
    // Display the decimal.FromOACurrency parameter and decimal result.
    public static void ShowDecimalFromOACurrency( long Argument )
    {
        decimal decCurrency = decimal.FromOACurrency( Argument );

        Console.WriteLine( dataFmt, Argument, decCurrency );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the " +
            "decimal.FromOACurrency( ) method generates \nthe " +
            "following output. It displays the OLE Automation " +
            "Currency \nvalue as a long and the result as a " +
            "decimal.\n" );
        Console.WriteLine( dataFmt, "OA Currency", "Decimal Value" );
        Console.WriteLine( dataFmt, "-----------", "-------------" );

        // Convert OLE Automation Currency values to decimal objects.
        ShowDecimalFromOACurrency( 0L );
        ShowDecimalFromOACurrency( 1L );
        ShowDecimalFromOACurrency( 100000L );
        ShowDecimalFromOACurrency( 100000000000L );
        ShowDecimalFromOACurrency( 1000000000000000000L );
        ShowDecimalFromOACurrency( 1000000000000000001L );
        ShowDecimalFromOACurrency( long.MaxValue );
        ShowDecimalFromOACurrency( long.MinValue );
        ShowDecimalFromOACurrency( 123456789L );
        ShowDecimalFromOACurrency( 1234567890000L );
        ShowDecimalFromOACurrency( 1234567890987654321 );
        ShowDecimalFromOACurrency( 4294967295L );
    }
}

/*
This example of the decimal.FromOACurrency( ) method generates
the following output. It displays the OLE Automation Currency
value as a long and the result as a decimal.

          OA Currency            Decimal Value
          -----------            -------------
                    0                        0
                    1                   0.0001
               100000                       10
         100000000000                 10000000
  1000000000000000000          100000000000000
  1000000000000000001     100000000000000.0001
  9223372036854775807     922337203685477.5807
 -9223372036854775808    -922337203685477.5808
            123456789               12345.6789
        1234567890000                123456789
  1234567890987654321     123456789098765.4321
          4294967295              429496.7295
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ToOACurrency(Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.GetBits Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetBits(Decimal)	Converts the value of a specified instance of Decimal to its equivalent binary representation.
GetBits(Decimal, Span<Int32>)	Converts the value of a specified instance of Decimal to its equivalent binary representation.

GetBits(Decimal)

Converts the value of a specified instance of Decimal to its equivalent binary representation.

C#

```
public static int[] GetBits (decimal d);
```

Parameters

d Decimal

The value to convert.

Returns

Int32[]

A 32-bit signed integer array with four elements that contain the binary representation of `d`.

Examples

The following example uses the `GetBits` method to convert several `Decimal` values to their equivalent binary representations. It then displays the decimal values and the hexadecimal value of the elements in the array returned by the GetBits method.

C#

```
using System;

class Example
{
   public static void Main()
   {
      // Define an array of Decimal values.
      Decimal[] values = { 1M, 100000000000000M,
1000000000000000000000000000M,
                           100000000000000.00000000000000M,
1.0000000000000000000000000000M,
                           123456789M, 0.123456789M,
0.000000000123456789M,
                           0.000000000000000000123456789M, 4294967295M,
                           18446744073709551615M, Decimal.MaxValue,
                           Decimal.MinValue,
-7.9228162514264337593543950335M };

      Console.WriteLine("{0,31}  {1,10:X8}{2,10:X8}{3,10:X8}{4,10:X8}",
                        "Argument", "Bits[3]", "Bits[2]", "Bits[1]",
                        "Bits[0]" );
      Console.WriteLine( "{0,31}  {1,10:X8}{2,10:X8}{3,10:X8}{4,10:X8}",
                         "--------", "-------", "-------", "-------",
                         "-------" );

      // Iterate each element and display its binary representation
      foreach (var value in values) {
        int[] bits = decimal.GetBits(value);
        Console.WriteLine("{0,31}  {1,10:X8}{2,10:X8}{3,10:X8}{4,10:X8}",
                          value, bits[3], bits[2], bits[1], bits[0]);
      }
   }
}
// The example displays the following output:
//                           Argument     Bits[3]   Bits[2]   Bits[1]
Bits[0]
//                           --------     -------   -------   -------   -
------
//                                  1    00000000  00000000  00000000
00000001
//                    100000000000000    00000000  00000000  00005AF3
107A4000
//      1000000000000000000000000000    00000000  204FCE5E  3E250261
10000000
//     100000000000000.00000000000000    000E0000  204FCE5E  3E250261
```

```
10000000
//     1.0000000000000000000000000000    001C0000  204FCE5E  3E250261
10000000
//                          123456789    00000000  00000000  00000000
075BCD15
//                        0.123456789    00090000  00000000  00000000
075BCD15
//               0.000000000123456789    00120000  00000000  00000000
075BCD15
//      0.000000000000000000123456789    001B0000  00000000  00000000
075BCD15
//                         4294967295    00000000  00000000  00000000
FFFFFFFF
//               18446744073709551615    00000000  00000000  FFFFFFFF
FFFFFFFF
//      79228162514264337593543950335    00000000  FFFFFFFF  FFFFFFFF
FFFFFFFF
//     -79228162514264337593543950335    80000000  FFFFFFFF  FFFFFFFF
FFFFFFFF
//    -7.9228162514264337593543950335    801C0000  FFFFFFFF  FFFFFFFF
FFFFFFFF
```

The following example uses the GetBits method to retrieve the component parts of an array. It then uses this array in the call to the Decimal(Int32, Int32, Int32, Boolean, Byte) constructor to instantiate a new Decimal value.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal[] values = { 1234.96m, -1234.96m };
      foreach (var value in values) {
         int[] parts = Decimal.GetBits(value);
         bool sign = (parts[3] & 0x80000000) != 0;

         byte scale = (byte) ((parts[3] >> 16) & 0x7F);
         Decimal newValue = new Decimal(parts[0], parts[1], parts[2],
sign, scale);
         Console.WriteLine("{0} --> {1}", value, newValue);
      }
   }
}
// The example displays the following output:
//       1234.96 --> 1234.96
//       -1234.96 --> -1234.96
```

Remarks

The binary representation of a Decimal number consists of a 1-bit sign, a 96-bit integer number, and a scaling factor used to divide the integer number and specify what portion of it is a decimal fraction. The scaling factor is implicitly the number 10, raised to an exponent ranging from 0 to 28.

The return value is a four-element array of 32-bit signed integers.

The first, second, and third elements of the returned array contain the low, middle, and high 32 bits of the 96-bit integer number.

The fourth element of the returned array contains the scale factor and sign. It consists of the following parts:

Bits 0 to 15, the lower word, are unused and must be zero.

Bits 16 to 23 must contain an exponent between 0 and 28, which indicates the power of 10 to divide the integer number.

Bits 24 to 30 are unused and must be zero.

Bit 31 contains the sign: 0 mean positive, and 1 means negative.

Note that the bit representation differentiates between negative and positive zero. These values are treated as being equal in all operations.

See also

- Decimal

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetBits(Decimal, Span<Int32>)

Converts the value of a specified instance of Decimal to its equivalent binary representation.

```
C#

public static int GetBits (decimal d, Span<int> destination);
```

Parameters

d Decimal

The value to convert.

destination Span<Int32>

The span into which to store the four-integer binary representation.

Returns

Int32

`4`, which is the number of integers in the binary representation.

Exceptions

ArgumentException

The destination span was not long enough to store the binary representation.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Decimal.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant Decimal.

Implements

GetTypeCode()

Examples

The following code example uses the `GetTypeCode` method to return the type code for `Decimal` value type.

C#

```
// Example of the decimal.GetTypeCode method.
using System;

class DecimalGetTypeCodeDemo
{
    public static void Main( )
    {
        Console.WriteLine( "This example of the " +
            "decimal.GetTypeCode( ) \nmethod " +
            "generates the following output.\n" );

        // Create a decimal object and get its type code.
```

```
        decimal aDecimal = new decimal( 1.0 );
        TypeCode typCode = aDecimal.GetTypeCode( );

        Console.WriteLine( "Type Code:      \"{0}\"", typCode );
        Console.WriteLine( "Numeric value:  {0}", (int)typCode );
    }
}

/*
This example of the decimal.GetTypeCode( )
method generates the following output.

Type Code:      "Decimal"
Numeric value:  15
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IsCanonical(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
public static bool IsCanonical (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IsEvenInteger(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IsInteger(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
public static bool IsInteger (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` and `3.0` will return `true` while `2.2` and `3.3` will return `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IsNegative(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IsOddInteger(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IsPositive(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Max(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static decimal Max (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.MaxMagnitude(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static decimal MaxMagnitude (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf>, this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Min(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
public static decimal Min (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.MinMagnitude(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
public static decimal MinMagnitude (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf>, this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Multiply(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two specified Decimal values.

C#

```
public static decimal Multiply (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The multiplicand.

d2 Decimal

The multiplier.

Returns

Decimal

The result of multiplying `d1` and `d2`.

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following code example creates several pairs of `Decimal` values and calculates their products with the `Multiply` method.

C#

```
// Example of the decimal.Multiply, decimal.Divide, and
// decimal.Remainder methods.
using System;
using Microsoft.VisualBasic;

class DecimalMulDivRemDemo
{
    const string dataFmt = "{0,-35}{1,31}";

    // Display decimal parameters and their product, quotient, and
    // remainder.
    public static void ShowDecimalProQuoRem( decimal Left, decimal Right )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "decimal Left", Left );
        Console.WriteLine( dataFmt, "decimal Right", Right );
        Console.WriteLine( dataFmt, "decimal.Multiply( Left, Right )",
            decimal.Multiply( Left, Right ) );
        Console.WriteLine( dataFmt, "decimal.Divide( Left, Right )",
            decimal.Divide( Left, Right ) );
        Console.WriteLine( dataFmt, "decimal.Remainder( Left, Right )",
            decimal.Remainder( Left, Right ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the \n" +
            "  decimal.Multiply( decimal, decimal ), \n" +
            "  decimal.Divide( decimal, decimal ), and \n" +
            "  decimal.Remainder( decimal, decimal ) \n" +
            "methods generates the following output. It displays " +
            "the product, \nquotient, and remainder of several " +
            "pairs of decimal objects." );

        // Create pairs of decimal objects.
        ShowDecimalProQuoRem( 1000M, 7M );
        ShowDecimalProQuoRem( -1000M, 7M );
        ShowDecimalProQuoRem(
            new decimal( 1230000000, 0, 0, false, 7 ), 0.0012300M );
        ShowDecimalProQuoRem( 12345678900000000M,
            0.0000000012345678M );
        ShowDecimalProQuoRem( 123456789.0123456789M,
            123456789.1123456789M );
    }
}

/*
This example of the
  decimal.Multiply( decimal, decimal ),
  decimal.Divide( decimal, decimal ), and
  decimal.Remainder( decimal, decimal )
methods generates the following output. It displays the product,
```

```
quotient, and remainder of several pairs of decimal objects.

decimal Left                                                  1000
decimal Right                                                    7
decimal.Multiply( Left, Right )                               7000
decimal.Divide( Left, Right )       142.857142857142857142857142 86
decimal.Remainder( Left, Right )                                 6

decimal Left                                                 -1000
decimal Right                                                    7
decimal.Multiply( Left, Right )                              -7000
decimal.Divide( Left, Right )      -142.857142857142857142857142 86
decimal.Remainder( Left, Right )                                -6

decimal Left                                            123.0000000
decimal Right                                             0.0012300
decimal.Multiply( Left, Right )                   0.15129000000000
decimal.Divide( Left, Right )                               100000
decimal.Remainder( Left, Right )                                 0

decimal Left                                       12345678900000000
decimal Right                                     0.0000000012345678
decimal.Multiply( Left, Right )          15241577.6390794200000000
decimal.Divide( Left, Right )       1000000072900005977800490 1.796
decimal.Remainder( Left, Right )                    0.000000000983

decimal Left                                  123456789.0123456789
decimal Right                                 123456789.1123456789
decimal.Multiply( Left, Right )     15241578765584515.651425087878
decimal.Divide( Left, Right )       0.999999999189999993366099944 9
decimal.Remainder( Left, Right )              123456789.0123456789
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(Decimal, Decimal)
- Subtract(Decimal, Decimal)

- Divide(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Negate(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the result of multiplying the specified Decimal value by negative one.

```
C#

public static decimal Negate (decimal d);
```

Parameters

d Decimal

The value to negate.

Returns

Decimal

A decimal number with the value of `d`, but the opposite sign.

-or-

Zero, if `d` is zero.

Examples

The following code example uses the `Negate` method to change the sign of several `Decimal` values.

```
C#

// Example of the decimal.Negate, decimal.Floor, and decimal.Truncate
// methods.
using System;

class DecimalFloorNegTruncDemo
{
```

```
    const string dataFmt = "{0,-30}{1,26}";

    // Display decimal parameters and the method results.
    public static void ShowDecimalFloorNegTrunc( decimal Argument )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "decimal Argument", Argument );
        Console.WriteLine( dataFmt, "decimal.Negate( Argument )",
            decimal.Negate( Argument ) );
        Console.WriteLine( dataFmt, "decimal.Floor( Argument )",
            decimal.Floor( Argument ) );
        Console.WriteLine( dataFmt, "decimal.Truncate( Argument )",
            decimal.Truncate( Argument ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the \n" +
            "  decimal.Negate( decimal ), \n" +
            "  decimal.Floor( decimal ), and \n" +
            "  decimal.Truncate( decimal ) \n" +
            "methods generates the following output." );

        // Create pairs of decimal objects.
        ShowDecimalFloorNegTrunc( 0M );
        ShowDecimalFloorNegTrunc( 123.456M );
        ShowDecimalFloorNegTrunc( -123.456M );
        ShowDecimalFloorNegTrunc(
            new decimal( 1230000000, 0, 0, true, 7 ) );
        ShowDecimalFloorNegTrunc( -9999999999.9999999999M );
    }
}

/*
This example of the
  decimal.Negate( decimal ),
  decimal.Floor( decimal ), and
  decimal.Truncate( decimal )
methods generates the following output.

decimal Argument                                    0
decimal.Negate( Argument )                          0
decimal.Floor( Argument )                           0
decimal.Truncate( Argument )                        0

decimal Argument                              123.456
decimal.Negate( Argument )                   -123.456
decimal.Floor( Argument )                         123
decimal.Truncate( Argument )                      123

decimal Argument                             -123.456
decimal.Negate( Argument )                    123.456
decimal.Floor( Argument )                        -124
decimal.Truncate( Argument )                     -123
```

```
decimal Argument                          -123.0000000
decimal.Negate( Argument )                 123.0000000
decimal.Floor( Argument )                         -123
decimal.Truncate( Argument )                      -123

decimal Argument                 -9999999999.9999999999
decimal.Negate( Argument )        9999999999.9999999999
decimal.Floor( Argument )                 -10000000000
decimal.Truncate( Argument )               -9999999999
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its Decimal equivalent.

Overloads

Expand table

Parse(String)	Converts the string representation of a number to its Decimal equivalent.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its Decimal equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number to its Decimal equivalent using the specified culture-specific format information.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format.

Parse(String)

Converts the string representation of a number to its Decimal equivalent.

```
C#

public static decimal Parse (string s);
```

Parameters

s String

The string representation of the number to convert.

Returns

Decimal

The equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following code example uses the Parse(String) method to parse string representations of Decimal values.

```
C#

string value;
decimal number;
// Parse an integer with thousands separators.
value = "16,523,421";
number = Decimal.Parse(value);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '16,523,421' converted to 16523421.
```

```
// Parse a floating point value with thousands separators
value = "25,162.1378";
number = Decimal.Parse(value);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '25,162.1378' converted to 25162.1378.

// Parse a floating point number with US currency symbol.
value = "$16,321,421.75";
try
{
   number = Decimal.Parse(value);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '$16,321,421.75'.

// Parse a number in exponential notation
value = "1.62345e-02";
try
{
   number = Decimal.Parse(value);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '1.62345e-02'.
```

Remarks

Parameter `s` contains a number of the form:

[ws][sign][digits,]digits[.fractional-digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.

Element	Description
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.
,	A culture-specific thousands separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A sequence of digits ranging from 0 to 9.

Parameter `s` is interpreted using the NumberStyles.Number style. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use either the Decimal.Parse(String, NumberStyles) or the Decimal.Parse(String, NumberStyles, IFormatProvider) method.

Parameter `s` is parsed using the formatting information in a NumberFormatInfo initialized for the current system culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other culture, use the Decimal.Parse(String, IFormatProvider) or Decimal.Parse(String, NumberStyles, IFormatProvider) method.

If necessary, the value of `s` is rounded using rounding to nearest.

A Decimal has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If during a parse operation a separator is encountered in the `s` parameter, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- TryParse
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static decimal Parse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Decimal

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static decimal Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Decimal

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its Decimal equivalent.

```
C#

public static decimal Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

The string representation of the number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the style elements that can be present in `s`. A typical value to specify is Number.

Returns

Decimal

The Decimal number equivalent to the number contained in `s` as specified by `style`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Decimal.MinValue or greater than Decimal.MaxValue

Examples

The following code example uses the Parse(String, NumberStyles) method to parse the string representations of Decimal values using the en-US culture.

C#

```
string value;
decimal number;
NumberStyles style;

// Parse string with a floating point value using NumberStyles.None.
value = "8694.12";
style = NumberStyles.None;
try
{
   number = Decimal.Parse(value, style);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '8694.12'.

// Parse string with a floating point value and allow decimal point.
style = NumberStyles.AllowDecimalPoint;
number = Decimal.Parse(value, style);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '8694.12' converted to 8694.12.

// Parse string with negative value in parentheses
value = "(1,789.34)";
style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowThousands |
        NumberStyles.AllowParentheses;
number = Decimal.Parse(value, style);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '(1,789.34)' converted to -1789.34.

// Parse string using Number style
value = " -17,623.49 ";
style = NumberStyles.Number;
number = Decimal.Parse(value, style);
Console.WriteLine("'{0}' converted to {1}.", value, number);
```

```
// Displays:
//    ' -17,623.49 ' converted to -17623.49.
```

Remarks

The `style` parameter defines the style elements (such as thousands separators, white space, and currency symbols) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- NumberStyles.AllowHexSpecifier
- HexNumber

Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional-digits][e[sign]digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits ranging from 0 to 9.
,	A culture-specific thousands separator symbol. The current culture's thousands separator can appear in `s` if `style` includes the NumberStyles.AllowThousands

Element	Description
	flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A sequence of digits ranging from 0 to 9. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the None style) always parses successfully if it is in the range of the Decimal type. The remaining NumberStyles members control elements that may be but are not required to be present in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The . and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation. This flag supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with elements such as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.

NumberStyles value	Elements permitted in s in addition to digits
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The , element.
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the `.` symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, `,` and `.` elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see CurrentInfo.

A Decimal has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If a separator is encountered in the `s` parameter during a parse operation, `styles` includes the NumberStyles.AllowThousands and NumberStyles.AllowDecimalPoint values, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- TryParse
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String, IFormatProvider)

Converts the string representation of a number to its Decimal equivalent using the specified culture-specific format information.

```
C#

public static decimal Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string representation of the number to convert.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific parsing information about `s`.

Returns

Decimal

The Decimal number equivalent to the number contained in `s` as specified by `provider`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to a Decimal value.

C#

```
protected void OkToDecimal_Click(object sender, EventArgs e)
{
    string locale;
    decimal number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Decimal.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
```

```
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

This overload of the Parse(String, IFormatProvider) method is commonly used to convert text that can be formatted in a variety of ways to a Decimal value. For example, it can be used to convert the text entered by a user into an HTML text box to a numeric value.

The `s` parameter contains a number of the form:

[ws][sign][digits,]digits[.fractional-digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.
,	A culture-specific thousands separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Number style. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use the Decimal.Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the thread current culture is used.

A Decimal object has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- TryParse
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```C#
public static decimal Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Number, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Decimal

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format.

```C#
public static decimal Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Number, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

The span containing the characters representing the number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the style elements that can be present in `s`. A typical value to specify is Number.

`provider` IFormatProvider

An IFormatProvider object that supplies culture-specific information about the format of `s`.

Returns

Decimal

The Decimal number equivalent to the number contained in `s` as specified by `style` and `provider`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format.

```
C#

public static decimal Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

The string representation of the number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the style elements that can be present in `s`. A typical value to specify is Number.

`provider` IFormatProvider

An IFormatProvider object that supplies culture-specific information about the format of `s`.

Returns

Decimal

The Decimal number equivalent to the number contained in `s` as specified by `style` and `provider`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Decimal.MinValue or greater than Decimal.MaxValue.

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

Examples

The following example uses a variety of `style` and `provider` parameters to parse the string representations of Decimal values.

C#

```
string value;
decimal number;
NumberStyles style;
CultureInfo provider;

// Parse string using " " as the thousands separator
// and "," as the decimal separator for fr-FR culture.
value = "892 694,12";
style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowThousands;
provider = new CultureInfo("fr-FR");

number = Decimal.Parse(value, style, provider);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '892 694,12' converted to 892694.12.

try
{
   number = Decimal.Parse(value, style, CultureInfo.InvariantCulture);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '892 694,12'.

// Parse string using "$" as the currency symbol for en-GB and
```

```
// en-US cultures.
value = "$6,032.51";
style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
provider = new CultureInfo("en-GB");

try
{
   number = Decimal.Parse(value, style, provider);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '$6,032.51'.

provider = new CultureInfo("en-US");
number = Decimal.Parse(value, style, provider);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '$6,032.51' converted to 6032.51.
```

Remarks

The `style` parameter defines the allowable format of the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- NumberStyles.AllowHexSpecifier
- HexNumber

Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional-digits][e[sign]digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyNegativePattern and CurrencyPositivePattern properties of the NumberFormatInfo object returned by the GetFormat method of the `provider`

Element	Description
	parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits ranging from 0 to 9.
,	A culture-specific thousands separator symbol. The thousands separator of the culture defined by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture defined by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A sequence of digits ranging from 0 to 9. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the None style) always parses successfully if it is in the range of the Decimal type. The remaining NumberStyles members control elements that may be but are not required to be present in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The . and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation. This flag supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with elements such as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The , element.
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the . symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, ,, and . elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the thread current culture is used.

A Decimal object has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse

operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- TryParse
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Remainder(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the remainder after dividing two Decimal values.

C#

```
public static decimal Remainder (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The dividend.

d2 Decimal

The divisor.

Returns

Decimal

The remainder after dividing `d1` by `d2`.

Exceptions

DivideByZeroException

`d2` is zero.

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example uses the Remainder method to calculate the remainder in a series of division operations.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        // Create parallel arrays of Decimals to use as the dividend and
divisor.
        Decimal[] dividends = { 79m, 1000m, -1000m, 123m, 1234567800000m,
                                1234.0123m };
        Decimal[] divisors = { 11m, 7m, 7m, .00123m, 0.12345678m, 1234.5678m
};

        for (int ctr = 0; ctr < dividends.Length; ctr++)
        {
           Decimal dividend = dividends[ctr];
           Decimal divisor = divisors[ctr];
           Console.WriteLine("{0:N3} / {1:N3} = {2:N3} Remainder {3:N3}",
dividend,
                             divisor, Decimal.Divide(dividend, divisor),
                             Decimal.Remainder(dividend, divisor));
        }
    }
}
// The example displays the following output:
//       79.000 / 11.000 = 7.182 Remainder 2.000
//       1,000.000 / 7.000 = 142.857 Remainder 6.000
//       -1,000.000 / 7.000 = -142.857 Remainder -6.000
//       123.000 / 0.001 = 100,000.000 Remainder 0.000
//       1,234,567,800,000.000 / 0.123 = 10,000,000,000,000.000 Remainder
0.000
//       1,234.012 / 1,234.568 = 1.000 Remainder 1,234.012
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rounds a value to the nearest integer or specified number of decimal places.

Overloads

Expand table

Round(Decimal, Int32, MidpointRounding)	Rounds a decimal value to the specified precision using the specified rounding strategy.
Round(Decimal, MidpointRounding)	Rounds a decimal value to an integer using the specified rounding strategy.
Round(Decimal, Int32)	Rounds a Decimal value to a specified number of decimal places.
Round(Decimal)	Rounds a decimal value to the nearest integer.

Round(Decimal, Int32, MidpointRounding)

Rounds a decimal value to the specified precision using the specified rounding strategy.

C#

```
public static decimal Round (decimal d, int decimals, MidpointRounding
mode);
```

Parameters

d Decimal

A decimal number to round.

`decimals` Int32

The number of significant decimal places (precision) in the return value.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Decimal

The number that `d` is rounded to using the `mode` rounding strategy and with a precision of `decimals`. If the precision of `d` is less than `decimals`, `d` is returned unchanged.

Implements

Round(TSelf, Int32, MidpointRounding)

Exceptions

ArgumentOutOfRangeException

`decimals` is less than 0 or greater than 28.

ArgumentException

`mode` is not a MidpointRounding value.

OverflowException

The result is outside the range of a Decimal object.

Examples

The following example demonstrates how to use the Round(Decimal, Int32, MidpointRounding) method with the MidpointRounding enumeration.

C#

```
decimal result;

// Round a positive value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
```

```
MidpointRounding.ToEven)");
result = Math.Round(3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(3.47m, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result} = Math.Round({3.47m}, 1,
MidpointRounding.ToZero)\n");

// Round a negative value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(-3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.ToEven)");
result = Math.Round(-3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(-3.47m, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result} = Math.Round({-3.47m}, 1,
MidpointRounding.ToZero)\n");

/*
This code example produces the following results:

3.4 = Math.Round(3.45, 1, MidpointRounding.ToEven)
3.5 = Math.Round(3.45, 1, MidpointRounding.AwayFromZero)
3.4 = Math.Round(3.47, 1, MidpointRounding.ToZero)

-3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
-3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
-3.4 = Math.Round(-3.47, 1, MidpointRounding.ToZero)
*/
```

Remarks

The `decimals` parameter specifies the number of significant decimal places in the return value and ranges from 0 to 28. If `decimals` is zero, an integer is returned.

If you specify ToEven or AwayFromZero for the `mode` parameter, those rounding strategies are only applied for midpoint values, that is, values whose least significant digit is 5.

See also

- Round(Decimal, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Round(Decimal, MidpointRounding)

Rounds a decimal value to an integer using the specified rounding strategy.

```
C#

public static decimal Round (decimal d, MidpointRounding mode);
```

Parameters

`d` Decimal

A decimal number to round.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Decimal

The integer that `d` is rounded to using the `mode` rounding strategy.

Implements

Round(TSelf, MidpointRounding)

Exceptions

ArgumentException

`mode` is not a MidpointRounding value.

OverflowException

The result is outside the range of a Decimal object.

Examples

The following example displays values returned by the Round(Decimal, MidpointRounding) method with different `mode` arguments.

C#

```
Console.WriteLine($"{"Value",-10} {"Default",-10} {"ToEven",-10}
{"AwayFromZero",-15} {"ToZero",-15}");
for (decimal value = 12.0m; value <= 13.0m; value += 0.1m)
    Console.WriteLine($"{value,-10} {Math.Round(value),-10} " +
        $"{Math.Round(value, MidpointRounding.ToEven),-10} " +
        $"{Math.Round(value, MidpointRounding.AwayFromZero),-15} " +
        $"{Math.Round(value, MidpointRounding.ToZero),-15}");

// The example displays the following output:
//       Value      Default    ToEven     AwayFromZero    ToZero
//       12.0       12         12         12              12
//       12.1       12         12         12              12
//       12.2       12         12         12              12
//       12.3       12         12         12              12
//       12.4       12         12         12              12
//       12.5       12         12         13              12
//       12.6       13         13         13              12
//       12.7       13         13         13              12
//       12.8       13         13         13              12
//       12.9       13         13         13              12
//       13.0       13         13         13              13
```

Remarks

If you specify ToEven or AwayFromZero for the `mode` parameter, those rounding strategies are only applied for midpoint values, that is, values whose least significant digit is 5.

See also

- Round(Decimal, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Round(Decimal, Int32)

Rounds a Decimal value to a specified number of decimal places.

```
C#
public static decimal Round (decimal d, int decimals);
```

Parameters

`d` Decimal

A decimal number to round.

`decimals` Int32

A value from 0 to 28 that specifies the number of decimal places to round to.

Returns

Decimal

The decimal number equivalent to `d` rounded to `decimals` decimal places.

Implements

Round(TSelf, Int32)

Exceptions

ArgumentOutOfRangeException

`decimals` is not a value from 0 to 28.

Examples

The following example rounds several `Decimal` values to a specified number of decimal places using the `Round` method.

C#

```
using System;

class Example12
{
   public static void Main()
   {
      // Define a set of Decimal values.
      decimal[] values = { 1.45m, 1.55m, 123.456789m, 123.456789m,
                           123.456789m, -123.456m,
                           new Decimal(1230000000, 0, 0, true, 7 ),
                           new Decimal(1230000000, 0, 0, true, 7 ),
                           -9999999999.9999999999m,
                           -9999999999.9999999999m };
      // Define a set of integers to for decimals argument.
      int[] decimals = { 1, 1, 4, 6, 8, 0, 3, 11, 9, 10};

      Console.WriteLine("{0,26}{1,8}{2,26}",
                        "Argument", "Digits", "Result" );
      Console.WriteLine("{0,26}{1,8}{2,26}",
                        "--------", "------", "------" );
      for (int ctr = 0; ctr < values.Length; ctr++)
        Console.WriteLine("{0,26}{1,8}{2,26}",
                          values[ctr], decimals[ctr],
                          Decimal.Round(values[ctr], decimals[ctr]));
    }
}
// The example displays the following output:
//                    Argument  Digits                    Result
//                    --------  ------                    ------
//                        1.45       1                       1.4
//                        1.55       1                       1.6
//                  123.456789       4                  123.4568
//                  123.456789       6                 123.456789
//                  123.456789       8                 123.456789
//                    -123.456       0                      -123
//                -123.0000000       3                  -123.000
//                -123.0000000      11               -123.0000000
//      -9999999999.9999999999       9    -10000000000.000000000
//      -9999999999.9999999999      10    -9999999999.9999999999
```

Remarks

This method is equivalent to calling the Round(Decimal, Int32, MidpointRounding) method with a `mode` argument of MidpointRounding.ToEven. When `d` is exactly halfway between two rounded values, the result is the rounded value that has an

even digit in the far right decimal position. For example, when rounded to two decimals, the value 2.345 becomes 2.34 and the value 2.355 becomes 2.36. This process is known as *rounding toward even*, or *banker's rounding*. It minimizes rounding errors that result from consistently rounding a midpoint value in a single direction.

See also

- Floor(Decimal)
- Truncate(Decimal)
- Round(Decimal, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Round(Decimal)

Rounds a decimal value to the nearest integer.

C#

```
public static decimal Round (decimal d);
```

Parameters

d Decimal

A decimal number to round.

Returns

Decimal

The integer that is nearest to the `d` parameter. If `d` is halfway between two integers, one of which is even and the other odd, the even number is returned.

Implements

Round(TSelf)

Exceptions

OverflowException

The result is outside the range of a Decimal value.

Examples

The following example rounds a range of Decimal values between 100 and 102 to the nearest integer. Because the method uses banker's rounding, 100.5 rounds to 100 and 101.5 rounds to 102.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      for (decimal value = 100m; value <= 102m; value += .1m)
         Console.WriteLine("{0} --> {1}", value, Decimal.Round(value));
   }
}
// The example displays the following output:
//     100 --> 100
//     100.1 --> 100
//     100.2 --> 100
//     100.3 --> 100
//     100.4 --> 100
//     100.5 --> 100
//     100.6 --> 101
//     100.7 --> 101
//     100.8 --> 101
//     100.9 --> 101
//     101.0 --> 101
//     101.1 --> 101
//     101.2 --> 101
//     101.3 --> 101
//     101.4 --> 101
//     101.5 --> 102
//     101.6 --> 102
//     101.7 --> 102
```

```
//     101.8 --> 102
//     101.9 --> 102
//     102.0 --> 102
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called *round half to even* or *banker's rounding*. It minimizes rounding errors that result from consistently rounding a midpoint value in a single direction. It is equivalent to calling the Round(Decimal, MidpointRounding) method with a `mode` argument of MidpointRounding.ToEven.

See also

- Round(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Sign(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (decimal d);
```

Parameters

d Decimal

The value whose sign is to be computed.

Returns

Int32

A positive value if `d` is positive, Zero if `d` is zero, and a negative value if `d` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Subtract(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts one specified Decimal value from another.

C#

```
public static decimal Subtract (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The minuend.

d2 Decimal

The subtrahend.

Returns

Decimal

The result of subtracting `d2` from `d1`.

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Examples

The following example illustrates the use of `Subtract`.

C#

```
class PiggyBank {
    public decimal Cents {
        get {
            return Decimal.Subtract(MyFortune, Decimal.Floor(MyFortune));
        }
    }

    protected decimal MyFortune;

    public void AddPenny() {
        MyFortune += .01m;
    }
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(Decimal, Decimal)
- Multiply(Decimal, Decimal)
- Divide(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToByte(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent 8-bit unsigned integer.

C#

```
public static byte ToByte (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Byte

An 8-bit unsigned integer equivalent to `value`.

Exceptions

OverflowException

`value` is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example uses the ToByte method to convert decimal numbers to Byte values.

C#

```
using System;

class Example
{
```

```
    public static void Main( )
    {
       decimal[] values = { 123m, new Decimal(78000, 0, 0, false, 3),
                            78.999m, 255.999m, 256m,
                            127.999m, 128m, -0.999m,
                            -1m,  -128.999m, -129m };

       foreach (var value in values) {
          try {
             byte number = Decimal.ToByte(value);
             Console.WriteLine("{0} --> {1}", value, number);
          }
          catch (OverflowException e)
          {
              Console.WriteLine("{0}: {1}", e.GetType().Name, value);
          }
       }
    }
 }
 // The example displays the following output:
 //     78 --> 78
 //     78.000 --> 78
 //     78.999 --> 78
 //     255.999 --> 255
 //     OverflowException: 256
 //     127.999 --> 127
 //     128 --> 128
 //     -0.999 --> 0
 //     OverflowException: -1
 //     OverflowException: -128.999
 //     OverflowException: -129
```

Remarks

Parameter `value` is rounded to the nearest integer value toward zero, and that result is returned.

You can also convert a Decimal value to an 8-bit unsigned integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Byte

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToDouble(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent double-precision floating-point number.

```
C#

public static double ToDouble (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

Double

A double-precision floating-point number equivalent to `d`.

Examples

The following code example converts `Decimal` numbers to Double values using `ToDouble` method.

```
C#

// Example of the decimal.ToSingle and decimal.ToDouble methods.
using System;

class DecimalToSgl_DblDemo
{
    static string formatter = "{0,30}{1,17}{2,23}";

    // Convert the decimal argument; no exceptions are thrown.
    public static void DecimalToSgl_Dbl( decimal argument )
```

```
    {
        object SingleValue;
        object DoubleValue;

        // Convert the argument to a float value.
        SingleValue = decimal.ToSingle( argument );

        // Convert the argument to a double value.
        DoubleValue = decimal.ToDouble( argument );

        Console.WriteLine( formatter, argument,
            SingleValue, DoubleValue );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the \n" +
            "  decimal.ToSingle( decimal ) and \n" +
            "  decimal.ToDouble( decimal ) \nmethods " +
            "generates the following output. It \ndisplays " +
            "several converted decimal values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "float", "double" );
        Console.WriteLine( formatter, "----------------",
            "-----", "------" );

        // Convert decimal values and display the results.
        DecimalToSgl_Dbl( 0.0000000000000000000000000001M );
        DecimalToSgl_Dbl( 0.0000000000123456789123456789M );
        DecimalToSgl_Dbl( 123M );
        DecimalToSgl_Dbl( new decimal( 123000000, 0, 0, false, 6 ) );
        DecimalToSgl_Dbl( 123456789.123456789M );
        DecimalToSgl_Dbl( 1234567891234567891234567 89M );
        DecimalToSgl_Dbl( decimal.MinValue );
        DecimalToSgl_Dbl( decimal.MaxValue );
    }
}

/*
This example of the
  decimal.ToSingle( decimal ) and
  decimal.ToDouble( decimal )
methods generates the following output. It
displays several converted decimal values.

               decimal argument            float                  double
               ----------------            -----                  ------
0.0000000000000000000000000001             1E-28                   1E-28
0.0000000000123456789123456789     1.234568E-11   1.23456789123457E-11
                            123              123                     123
                     123.000000              123                     123
           123456789.123456789     1.234568E+08       123456789.123457
   1234567891234567891234567 89     1.234568E+26   1.23456789123457E+26
-79228162514264337593543950335    -7.922816E+28  -7.92281625142643E+28
```

```
 79228162514264337593543950335     7.922816E+28   7.92281625142643E+28
*/
```

Remarks

This operation can produce round-off errors because a double-precision floating-point number has fewer significant digits than a Decimal.

You can also convert a Decimal to a Double value by using the Explicit assignment operator. Because the conversion can entail a loss of precision, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToInt16(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent 16-bit signed integer.

C#

```
public static short ToInt16 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Int16

A 16-bit signed integer equivalent to `value`.

Exceptions

OverflowException

`value` is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example uses the ToInt16 method to convert decimal numbers to Int16 values.

C#

```
using System;

class Example
{
```

```
    public static void Main( )
    {
       decimal[] values = { 123m, new Decimal(123000, 0, 0, false, 3),
                            123.999m, 65535.999m, 65536m,
                            32767.999m, 32768m, -0.999m,
                            -1m,  -32768.999m, -32769m };

       foreach (var value in values) {
          try {
             short number = Decimal.ToInt16(value);
             Console.WriteLine("{0} --> {1}", value, number);
          }
          catch (OverflowException e)
          {
              Console.WriteLine("{0}: {1}", e.GetType().Name, value);
          }
       }
    }
 }
 // The example displays the following output:
 //     123 --> 123
 //     123.000 --> 123
 //     123.999 --> 123
 //     OverflowException: 65535.999
 //     OverflowException: 65536
 //     32767.999 --> 32767
 //     OverflowException: 32768
 //     -0.999 --> 0
 //     -1 --> -1
 //     -32768.999 --> -32768
 //     OverflowException: -32769
```

Remarks

You can also convert a Decimal value to a 16-bit integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int16

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToInt32(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent 32-bit signed integer.

C#

```
public static int ToInt32 (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

Int32

A 32-bit signed integer equivalent to the value of `d`.

Exceptions

OverflowException

`d` is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example uses the ToInt32 method to convert decimal numbers to Int32 values.

C#

```
using System;

class Example
{
```

```
    public static void Main( )
    {
       decimal[] values = { 123m, new decimal(123000, 0, 0, false, 3),
                            123.999m, 4294967295.999m, 4294967296m,
                            4294967296m, 2147483647.999m, 2147483648m,
                            -0.999m, -1m, -2147483648.999m, -2147483649m };

       foreach (var value in values) {
          try {
             int number = Decimal.ToInt32(value);
             Console.WriteLine("{0} --> {1}", value, number);
          }
          catch (OverflowException e)
          {
              Console.WriteLine("{0}: {1}", e.GetType().Name, value);
          }
       }
    }
 }
 // The example displays the following output:
 //      123 --> 123
 //      123.000 --> 123
 //      123.999 --> 123
 //      OverflowException: 4294967295.999
 //      OverflowException: 4294967296
 //      OverflowException: 4294967296
 //      2147483647.999 --> 2147483647
 //      OverflowException: 2147483648
 //      -0.999 --> 0
 //      -1 --> -1
 //      -2147483648.999 --> -2147483648
 //      OverflowException: -2147483649
```

Remarks

The return value is the integral part of the decimal value; fractional digits are truncated.

You can also convert a Decimal value to a 32-bit integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int32

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToInt64(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent 64-bit signed integer.

C#

```
public static long ToInt64 (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

Int64

A 64-bit signed integer equivalent to the value of `d`.

Exceptions

OverflowException

`d` is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example uses the ToInt64 method to convert decimal numbers to Int64 values.

C#

```
using System;

class Example
{
```

```
    public static void Main( )
    {
       decimal[] values = { 123m, new Decimal(123000, 0, 0, false, 3),
                            123.999m, 18446744073709551615.999m,
                            18446744073709551616m, 9223372036854775807.999m,
                            9223372036854775808m, -0.999m, -1m,
                            -9223372036854775808.999m,
                            -9223372036854775809m };

       foreach (var value in values) {
          try {
             long number = Decimal.ToInt64(value);
             Console.WriteLine("{0} --> {1}", value, number);
          }
          catch (OverflowException e)
          {
              Console.WriteLine("{0}: {1}", e.GetType().Name, value);
          }
       }
    }
 }
 // The example displays the following output:
 //   123 --> 123
 //   123.000 --> 123
 //   123.999 --> 123
 //   OverflowException: 18446744073709551615.999
 //   OverflowException: 18446744073709551616
 //   9223372036854775807.999 --> 9223372036854775807
 //   OverflowException: 9223372036854775808
 //   -0.999 --> 0
 //   -1 --> -1
 //   -9223372036854775808.999 --> -9223372036854775808
 //   OverflowException: -9223372036854775809
```

Remarks

The return value is the integral part of the decimal value; fractional digits are truncated.

You can also convert a Decimal value to a 64-bit integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToOACurrency(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified Decimal value to the equivalent OLE Automation Currency value, which is contained in a 64-bit signed integer.

```
C#

public static long ToOACurrency (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

Int64

A 64-bit signed integer that contains the OLE Automation equivalent of `value`.

Examples

The following code example uses the `ToOACurrency` method to convert `Decimal` numbers to the equivalent OLE Automation Currency values that are contained in Int64 fields.

```
C#

// Example of the decimal.ToOACurrency method.
using System;

class DecimalToOACurrencyDemo
{
    const string dataFmt = "{0,31}{1,27}";
```

```
    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Display the decimal.ToOACurrency parameter and the result
    // or exception.
    public static void ShowDecimalToOACurrency( decimal Argument )
    {
        // Catch the exception if ToOACurrency( ) throws one.
        try
        {
            long oaCurrency = decimal.ToOACurrency( Argument );
            Console.WriteLine( dataFmt, Argument, oaCurrency );
        }
        catch( Exception ex )
        {
            Console.WriteLine( dataFmt, Argument,
                GetExceptionType( ex ) );
        }
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the " +
            "decimal.ToOACurrency( ) method generates \nthe " +
            "following output. It displays the argument as a " +
            "decimal \nand the OLE Automation Currency value " +
            "as a long.\n" );
        Console.WriteLine( dataFmt, "Argument",
            "OA Currency or Exception" );
        Console.WriteLine( dataFmt, "--------",
            "------------------------" );

        // Convert decimal values to OLE Automation Currency values.
        ShowDecimalToOACurrency( 0M );
        ShowDecimalToOACurrency( 1M );
        ShowDecimalToOACurrency( 1.0000000000000000000000000000M );
        ShowDecimalToOACurrency( 100000000000000M );
        ShowDecimalToOACurrency( 100000000000000.00000000000000M );
        ShowDecimalToOACurrency( 1000000000000000000000000000M );
        ShowDecimalToOACurrency( 0.000000000123456789M );
        ShowDecimalToOACurrency( 0.123456789M );
        ShowDecimalToOACurrency( 123456789M );
        ShowDecimalToOACurrency( 123456789000000000M );
        ShowDecimalToOACurrency( 4294967295M );
        ShowDecimalToOACurrency( 18446744073709551615M );
        ShowDecimalToOACurrency( -79.228162514264337593543950335M );
        ShowDecimalToOACurrency( -79228162514264.337593543950335M );
    }
}
```

```
/*
This example of the decimal.ToOACurrency( ) method generates
the following output. It displays the argument as a decimal
and the OLE Automation Currency value as a long.

                        Argument   OA Currency or Exception
                        --------   ------------------------
                               0                          0
                               1                      10000
 1.0000000000000000000000000000                       10000
                 100000000000000        1000000000000000000
 100000000000000.00000000000000         1000000000000000000
  1000000000000000000000000000           OverflowException
           0.000000000123456789                          0
                    0.123456789                       1235
                      123456789               1234567890000
             123456789000000000           OverflowException
                     4294967295              42949672950000
           18446744073709551615           OverflowException
-79.228162514264337593543950335                     -792282
-79228162514264.337593543950335        -792281625142643376
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FromOACurrency(Int64)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.ToSByte(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.ToInt16(Decimal)**

Converts the value of the specified Decimal to the equivalent 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static sbyte ToSByte (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

SByte

An 8-bit signed integer equivalent to `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example uses the ToSByte method to convert decimal numbers to SByte values.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      decimal[] values = { 123m, new Decimal(78000, 0, 0, false, 3),
                           78.999m, 255.999m, 256m,
                           127.999m, 128m, -0.999m,
                           -1m,  -128.999m, -129m };

      foreach (var value in values) {
         try {
            sbyte number = Decimal.ToSByte(value);
            Console.WriteLine("{0} --> {1}", value, number);
         }
         catch (OverflowException e)
         {
             Console.WriteLine("{0}: {1}", e.GetType().Name, value);
         }
      }
   }
}
// The example displays the following output:
//     78 --> 78
//     78.000 --> 78
//     78.999 --> 78
//     OverflowException: 255.999
//     OverflowException: 256
//     127.999 --> 127
//     OverflowException: 128
//     -0.999 --> 0
//     -1 --> -1
//     -128.999 --> -128
//     OverflowException: -129
```

Remarks

You can also convert a Decimal value to an 8-bit signed integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- SByte

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToSingle(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the specified Decimal to the equivalent single-precision floating-point number.

C#

```
public static float ToSingle (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

Single

A single-precision floating-point number equivalent to the value of `d`.

Examples

The following code example converts `Decimal` numbers to Single values using the `ToSingle` method.

C#

```
// Example of the decimal.ToSingle and decimal.ToDouble methods.
using System;

class DecimalToSgl_DblDemo
{
    static string formatter = "{0,30}{1,17}{2,23}";

    // Convert the decimal argument; no exceptions are thrown.
    public static void DecimalToSgl_Dbl( decimal argument )
```

```
    {
        object SingleValue;
        object DoubleValue;

        // Convert the argument to a float value.
        SingleValue = decimal.ToSingle( argument );

        // Convert the argument to a double value.
        DoubleValue = decimal.ToDouble( argument );

        Console.WriteLine( formatter, argument,
            SingleValue, DoubleValue );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the \n" +
            "  decimal.ToSingle( decimal ) and \n" +
            "  decimal.ToDouble( decimal ) \nmethods " +
            "generates the following output. It \ndisplays " +
            "several converted decimal values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "float", "double" );
        Console.WriteLine( formatter, "----------------",
            "-----", "------" );

        // Convert decimal values and display the results.
        DecimalToSgl_Dbl( 0.0000000000000000000000000001M );
        DecimalToSgl_Dbl( 0.0000000000123456789123456789M );
        DecimalToSgl_Dbl( 123M );
        DecimalToSgl_Dbl( new decimal( 123000000, 0, 0, false, 6 ) );
        DecimalToSgl_Dbl( 123456789.123456789M );
        DecimalToSgl_Dbl( 1234567891234567891234567 89M );
        DecimalToSgl_Dbl( decimal.MinValue );
        DecimalToSgl_Dbl( decimal.MaxValue );
    }
}

/*
This example of the
  decimal.ToSingle( decimal ) and
  decimal.ToDouble( decimal )
methods generates the following output. It
displays several converted decimal values.

               decimal argument            float                  double
               ----------------            -----                  ------
0.0000000000000000000000000001             1E-28                   1E-28
0.0000000000123456789123456789     1.234568E-11   1.23456789123457E-11
                            123              123                     123
                     123.000000              123                     123
           123456789.123456789     1.234568E+08       123456789.123457
   1234567891234567891234567 89     1.234568E+26   1.23456789123457E+26
-79228162514264337593543950335    -7.922816E+28  -7.92281625142643E+28
```

```
 79228162514264337593543950335     7.922816E+28   7.92281625142643E+28
*/
```

Remarks

This operation can produce round-off errors because a single-precision floating-point number has fewer significant digits than a Decimal.

You can also convert a Decimal to a Single value by using the Explicit assignment operator. Because the conversion can entail a loss of precision, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Single

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent String representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

A string that represents the value of this instance.

Examples

The following example displays a Decimal value using the default ToString() method. It also displays the string representations of the Decimal value that result from using a number of standard format specifiers.

C#

```
decimal value = -16325.62m;
// Display value using default ToString method.
Console.WriteLine(value.ToString());            // Displays -16325.62
// Display value using some standard format specifiers.
Console.WriteLine(value.ToString("G"));         // Displays -16325.62
Console.WriteLine(value.ToString("C"));         // Displays ($16,325.62)
Console.WriteLine(value.ToString("F"));         // Displays -16325.62
```

The following example displays the amount of money in an account.

C#

```
class PiggyBank {
    public void AddPenny() {
        MyFortune = Decimal.Add(MyFortune, .01m);
    }

    public override string ToString() {
        return MyFortune.ToString("C")+" in piggy bank";
    }

    protected decimal MyFortune;
}
```

Remarks

The ToString() method formats a Decimal value in the default ("G", or general) format of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example displays the string representation of a Decimal value using CultureInfo objects that represent several different cultures.

C#

```
decimal value = -16325.62m;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));
// This example displays the following output to the console:
//       -16325.62
//       -16325.62
//       -16325,62
```

Remarks

The ToString(IFormatProvider) method formats a Decimal value in the default ("G", or general) format of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose IFormatProvider.GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a NumberFormatInfo object or a CultureInfo object. The `provider` parameter supplies culture-specific information used in formatting. If `provider` is `null`, the thread current culture is used.

To convert a Decimal value to its string representation using a specified culture and a specific format string, call the Decimal.ToString(String, IFormatProvider) method.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

format String

A standard or custom numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid.

Examples

The following example displays a Decimal value using each of the supported standard numeric format specifiers, together with two custom numeric format strings. In converting the numeric values to strings, the example uses the formatting conventions of the en-US culture.

```
C#

decimal value = 16325.62m;
string specifier;

// Use standard numeric format specifiers.
specifier = "G";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    G: 16325.62
specifier = "C";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    C: $16,325.62
specifier = "E04";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    E04: 1.6326E+004
specifier = "F";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    F: 16325.62
specifier = "N";
```

```
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    N: 16,325.62
specifier = "P";
Console.WriteLine("{0}: {1}", specifier,
(value/10000).ToString(specifier));
// Displays:    P: 163.26 %

// Use custom numeric format specifiers.
specifier = "0,0.000";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    0,0.000: 16,325.620
specifier = "#,#.00#;(#,#.00#)";
Console.WriteLine("{0}: {1}", specifier, (value*-1).ToString(specifier));
// Displays:    #,#.00#;(#,#.00#): (16,325.62)
```

Remarks

The ToString(String) method formats a Decimal value in a specified format by using the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The ToString method uses the standard or custom numeric format string specified by the `format` parameter to convert the value of the current instance into its string representation. The `format` parameter can be any valid standard numeric format specifier except for D, R, and X, as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string, the return value of this instance is formatted with the general numeric format specifier (G).

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The return value is formatted by using the NumberFormatInfo object for the current culture. To apply the formatting conventions of a specified culture, call the ToString(String, IFormatProvider) method.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid.

Examples

The following example displays a Decimal value using each of the supported standard numeric format specifiers for several different cultures.

C#

```
decimal value = 16325.62m;
string specifier;
CultureInfo culture;

// Use standard numeric format specifiers.
specifier = "G";
culture = CultureInfo.CreateSpecificCulture("eu-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,62
Console.WriteLine(value.ToString(specifier,
CultureInfo.InvariantCulture));
// Displays:    16325.62

specifier = "C";
culture = CultureInfo.CreateSpecificCulture("en-US");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    $16,325.62
culture = CultureInfo.CreateSpecificCulture("en-GB");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    £16,325.62
```

```
specifier = "E04";
culture = CultureInfo.CreateSpecificCulture("sv-SE");
Console.WriteLine(value.ToString(specifier, culture));
// Displays: 1,6326E+004
 culture = CultureInfo.CreateSpecificCulture("en-NZ");
 Console.WriteLine(value.ToString(specifier, culture));
// Displays:    1.6326E+004

specifier = "F";
culture = CultureInfo.CreateSpecificCulture("fr-FR");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,62
culture = CultureInfo.CreateSpecificCulture("en-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325.62

specifier = "N";
culture = CultureInfo.CreateSpecificCulture("es-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16.325,62
culture = CultureInfo.CreateSpecificCulture("fr-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16 325,62

specifier = "P";
culture = CultureInfo.InvariantCulture;
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.26 %
culture = CultureInfo.CreateSpecificCulture("ar-EG");
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.256 %
```

Remarks

The ToString(String, IFormatProvider) method formats a Decimal value in a specified format of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be any valid standard numeric format specifier except for D, R, and X, as well as any combination of custom numeric format specifiers. If

`format` is `null` or an empty string, the return value for this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose IFormatProvider.GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in formatting. If `provider` is `null`, the thread current culture is used.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.ToUInt16(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.ToInt32(Decimal)**

Converts the value of the specified Decimal to the equivalent 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ushort ToUInt16 (decimal value);
```

Parameters

`value` Decimal

The decimal number to convert.

Returns

UInt16

A 16-bit unsigned integer equivalent to the value of `value`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is greater than UInt16.MaxValue or less than UInt16.MinValue.

Examples

The following example uses the ToUInt16 method to convert decimal numbers to UInt16 values.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      decimal[] values = { 123m, new Decimal(123000, 0, 0, false, 3),
                           123.999m, 65535.999m, 65536m,
                           32767.999m, 32768m, -0.999m,
                           -1m,  -32768.999m, -32769m };

      foreach (var value in values) {
         try {
            ushort number = Decimal.ToUInt16(value);
            Console.WriteLine("{0} --> {1}", value, number);
         }
         catch (OverflowException e)
         {
             Console.WriteLine("{0}: {1}", e.GetType().Name, value);
         }
      }
   }
}
// The example displays the following output:
//     123 --> 123
//     123.000 --> 123
//     123.999 --> 123
//     65535.999 --> 65535
//     OverflowException: 65536
//     32767.999 --> 32767
//     32768 --> 32768
//     -0.999 --> 0
//     OverflowException: -1
//     OverflowException: -32768.999
//     OverflowException: -32769
```

Remarks

The return value is the integral part of the decimal value; fractional digits are truncated.

You can also convert a Decimal value to a 16-bit unsigned integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must

use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt16

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToUInt32(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.ToInt64(Decimal)**

Converts the value of the specified Decimal to the equivalent 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static uint ToUInt32 (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

UInt32

A 32-bit unsigned integer equivalent to the value of `d`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`d` is negative or greater than UInt32.MaxValue.

Examples

The following example uses the ToUInt32 method to convert decimal numbers to UInt32 values.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      decimal[] values = { 123m, new decimal(123000, 0, 0, false, 3),
                           123.999m, 4294967295.999m, 4294967296m,
                           4294967296m, 2147483647.999m, 2147483648m,
                           -0.999m, -1m, -2147483648.999m, -2147483649m };

      foreach (var value in values) {
         try {
            uint number = Decimal.ToUInt32(value);
            Console.WriteLine("{0} --> {1}", value, number);
         }
         catch (OverflowException e)
         {
             Console.WriteLine("{0}: {1}", e.GetType().Name, value);
         }
      }
   }
}
// The example displays the following output:
//     123 --> 123
//     123.000 --> 123
//     123.999 --> 123
//     4294967295.999 --> 4294967295
//     OverflowException: 4294967296
//     OverflowException: 4294967296
//     2147483647.999 --> 2147483647
//     2147483648 --> 2147483648
//     -0.999 --> 0
//     OverflowException: -1
//     OverflowException: -2147483648.999
//     OverflowException: -2147483649
```

Remarks

The return value is the integral part of the decimal value; fractional digits are truncated.

You can also convert a Decimal value to an 32-bit unsigned integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt32

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.ToUInt64(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified Decimal to the equivalent 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static ulong ToUInt64 (decimal d);
```

Parameters

d Decimal

The decimal number to convert.

Returns

UInt64

A 64-bit unsigned integer equivalent to the value of `d`.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`d` is negative or greater than UInt64.MaxValue.

Examples

The following example uses the ToUInt64 method to convert decimal numbers to UInt64 values.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      decimal[] values = { 123m, new Decimal(123000, 0, 0, false, 3),
                           123.999m, 18446744073709551615.999m,
                           18446744073709551616m, 9223372036854775807.999m,
                           9223372036854775808m, -0.999m, -1m,
                           -9223372036854775808.999m,
                           -9223372036854775809m };

      foreach (var value in values) {
         try {
            ulong number = Decimal.ToUInt64(value);
            Console.WriteLine("{0} --> {1}", value, number);
         }
         catch (OverflowException e)
         {
             Console.WriteLine("{0}: {1}", e.GetType().Name, value);
         }
      }
   }
}
// The example displays the following output:
//     123 --> 123
//     123.000 --> 123
//     123.999 --> 123
//     18446744073709551615.999 --> 18446744073709551615
//     OverflowException: 18446744073709551616
//     9223372036854775807.999 --> 9223372036854775807
//     9223372036854775808 --> 9223372036854775808
//     -0.999 --> 0
//     OverflowException: -1
//     OverflowException: -9223372036854775808.999
//     OverflowException: -9223372036854775809
```

Remarks

The return value is the integral part of the decimal value; fractional digits are truncated.

You can also convert a Decimal value to an 64-bit unsigned integer by using the Explicit assignment operator. Because the operator performs a narrowing conversion, you must use a casting operator in C# or a conversion function in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- UInt64

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Truncate(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the integral digits of the specified Decimal; any fractional digits are discarded.

C#

```
public static decimal Truncate (decimal d);
```

Parameters

d Decimal

The decimal number to truncate.

Returns

Decimal

The result of `d` rounded toward zero, to the nearest whole number.

Implements

Truncate(TSelf)

Examples

The following code example uses the `Truncate` method to discard the fractional digits of several `Decimal` values.

C#

```
// Example of the decimal.Negate, decimal.Floor, and decimal.Truncate
// methods.
using System;

class DecimalFloorNegTruncDemo
```

```
{
    const string dataFmt = "{0,-30}{1,26}";

    // Display decimal parameters and the method results.
    public static void ShowDecimalFloorNegTrunc( decimal Argument )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "decimal Argument", Argument );
        Console.WriteLine( dataFmt, "decimal.Negate( Argument )",
            decimal.Negate( Argument ) );
        Console.WriteLine( dataFmt, "decimal.Floor( Argument )",
            decimal.Floor( Argument ) );
        Console.WriteLine( dataFmt, "decimal.Truncate( Argument )",
            decimal.Truncate( Argument ) );
    }

    public static void Main( )
    {
        Console.WriteLine( "This example of the \n" +
            "  decimal.Negate( decimal ), \n" +
            "  decimal.Floor( decimal ), and \n" +
            "  decimal.Truncate( decimal ) \n" +
            "methods generates the following output." );

        // Create pairs of decimal objects.
        ShowDecimalFloorNegTrunc( 0M );
        ShowDecimalFloorNegTrunc( 123.456M );
        ShowDecimalFloorNegTrunc( -123.456M );
        ShowDecimalFloorNegTrunc(
            new decimal( 1230000000, 0, 0, true, 7 ) );
        ShowDecimalFloorNegTrunc( -9999999999.9999999999M );
    }
}

/*
This example of the
  decimal.Negate( decimal ),
  decimal.Floor( decimal ), and
  decimal.Truncate( decimal )
methods generates the following output.

decimal Argument                                    0
decimal.Negate( Argument )                          0
decimal.Floor( Argument )                           0
decimal.Truncate( Argument )                        0

decimal Argument                              123.456
decimal.Negate( Argument )                   -123.456
decimal.Floor( Argument )                         123
decimal.Truncate( Argument )                      123

decimal Argument                             -123.456
decimal.Negate( Argument )                    123.456
decimal.Floor( Argument )                        -124
decimal.Truncate( Argument )                     -123
```

```
decimal Argument                            -123.0000000
decimal.Negate( Argument )                   123.0000000
decimal.Floor( Argument )                           -123
decimal.Truncate( Argument )                        -123

decimal Argument                  -9999999999.9999999999
decimal.Negate( Argument )         9999999999.9999999999
decimal.Floor( Argument )                  -10000000000
decimal.Truncate( Argument )                -9999999999
*/
```

Remarks

This method rounds `d` toward zero, to the nearest whole number, which corresponds to discarding any digits after the decimal point.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Round(Decimal, Int32)
- Floor(Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current decimal instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current decimal instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

```
C#
```

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.TryGetBits(Decimal, Span<Int32>, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the value of a specified instance of Decimal to its equivalent binary representation.

C#

```
public static bool TryGetBits (decimal d, Span<int> destination, out int valuesWritten);
```

Parameters

d Decimal

The value to convert.

destination Span<Int32>

The span into which to store the binary representation.

valuesWritten Int32

When this method returns, contains the number of integers written to the destination.

Returns

Boolean

`true` if the decimal's binary representation was written to the destination; `false` if the destination wasn't long enough.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its Decimal equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, Decimal)	Tries to convert a UTF-8 character span containing the string representation of a number to its signed decimal equivalent.
TryParse(ReadOnlySpan<Char>, Decimal)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Decimal)	Converts the string representation of a number to its Decimal equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Decimal)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Decimal)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Decimal)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Decimal)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Decimal)	Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

TryParse(String, NumberStyles, IFormatProvider, Decimal)	Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, Decimal)

Tries to convert a UTF-8 character span containing the string representation of a number to its signed decimal equivalent.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text, out decimal
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Decimal

When this method returns, contains the signed decimal value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Decimal)

Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out decimal result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

result Decimal

When this method returns, contains the Decimal number that is equivalent to the numeric value contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not a number in a format compliant with `style`, or represents a number less than Decimal.MinValue or greater than Decimal.MaxValue. This parameter is passed uininitialized; any value originally supplied in `result` is overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Decimal)

Converts the string representation of a number to its Decimal equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (string? s, out decimal result);
```

Parameters

`s` String

The string representation of the number to convert.

`result` Decimal

When this method returns, contains the Decimal number that is equivalent to the numeric value contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not a number in a valid format, or represents a number less than Decimal.MinValue or greater than Decimal.MaxValue. This parameter is passed uininitialized; any value originally supplied in `result` is overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example uses the Decimal.TryParse(String, Decimal) method to convert the string representations of numeric values to Decimal values. It assumes that en-US is the current culture.

```
C#

string value;
decimal number;

// Parse a floating-point value with a thousands separator.
value = "1,643.57";
if (Decimal.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);
```

```
// Parse a floating-point value with a currency symbol and a
// thousands separator.
value = "$1,643.57";
if (Decimal.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);

// Parse value in exponential notation.
value = "-1.643e6";
if (Decimal.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);

// Parse a negative integer value.
value = "-1689346178821";
if (Decimal.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);
// The example displays the following output to the console:
//       1643.57
//       Unable to parse '$1,643.57'.
//       Unable to parse '-1.643e6'.
//       -1689346178821
```

Remarks

This overload differs from the Decimal.Parse(String) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed numeric value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

Parameter `s` contains a number of the form:

[ws][sign][digits,]digits[.fractional-digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.

Element	Description
digits	A sequence of digits ranging from 0 to 9.
,	A culture-specific thousands separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A sequence of digits ranging from 0 to 9.

Parameter `s` is interpreted using the NumberStyles.Number style. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use the Decimal.TryParse(String, NumberStyles, IFormatProvider, Decimal) method overload.

Parameter `s` is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other specified culture, use the Decimal.TryParse(String, NumberStyles, IFormatProvider, Decimal) method overload.

If necessary, the value of `s` is rounded using rounding to nearest.

A Decimal object has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If during a parse operation a separator is encountered in the `s` parameter, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- Parse
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Decimal)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out decimal result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Decimal

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, Decimal)

Tries to parse a span of characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out decimal result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Decimal

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Decimal)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
decimal result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Decimal

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Decimal)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
decimal result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Decimal

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Decimal)

Converts the span representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
decimal result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Number.

`provider` IFormatProvider

An object that supplies culture-specific parsing information about `s`.

`result` Decimal

When this method returns, contains the Decimal number that is equivalent to the numeric value contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not a number in a format compliant with `style`, or represents a number less than Decimal.MinValue or greater than Decimal.MaxValue. This parameter is passed uininitialized; any value originally supplied in `result` is overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Decimal)

Converts the string representation of a number to its Decimal equivalent using the specified style and culture-specific format. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out decimal result);
```

Parameters

`s` String

The string representation of the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Number.

`provider` IFormatProvider

An object that supplies culture-specific parsing information about `s`.

`result` Decimal

When this method returns, contains the Decimal number that is equivalent to the numeric value contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not a number in a format compliant with `style`, or represents a number less than Decimal.MinValue or greater than Decimal.MaxValue. This parameter is passed uininitialized; any value originally supplied in `result` is overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

Examples

The following example demonstrates the use of the TryParse(String, NumberStyles, IFormatProvider, Decimal) method to parse the string representation of a number that has a particular style and is formatted using the conventions of a particular culture.

C#

```
string value;
NumberStyles style;
CultureInfo culture;
decimal number;

// Parse currency value using en-GB culture.
value = "£1,097.63";
style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
culture = CultureInfo.CreateSpecificCulture("en-GB");
if (Decimal.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '£1,097.63' to 1097.63.

value = "1345,978";
style = NumberStyles.AllowDecimalPoint;
culture = CultureInfo.CreateSpecificCulture("fr-FR");
if (Decimal.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '1345,978' to 1345.978.
```

```
value = "1.345,978";
style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowThousands;
culture = CultureInfo.CreateSpecificCulture("es-ES");
if (Decimal.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '1.345,978' to 1345.978.

value = "1 345,978";
if (Decimal.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Unable to convert '1 345,978'.
```

Remarks

This overload differs from the Decimal.Parse(String, NumberStyles, IFormatProvider) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed numeric value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `style` parameter defines the allowable format of the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- NumberStyles.AllowHexSpecifier
- NumberStyles.HexNumber

Depending on the value of style, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional-digits][e[sign]digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag. It can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern or NumberFormatInfo.CurrencyPositivePattern properties of the NumberFormatInfo object returned by the IFormatProvider.GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.
.	A culture-specific decimal point symbol.
fractional-digits	A sequence of digits ranging from 0 to 9.

The `style` parameter specifies the permitted format of the `s` parameter, and can be one or more NumberStyles enumerated constants combined using a bitwise OR operation. If `style` is null, `s` is interpreted using the NumberStyles.Number style.

The `provider` parameter is an IFormatProvider implementation such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the thread current culture is used.

A Decimal object has 29 digits of precision. If `s` represents a number that has more than 29 digits, but has a fractional part and is within the range of MaxValue and MinValue, the number is rounded, not truncated, to 29 digits using rounding to nearest.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- Parse

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Addition(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two specified Decimal values.

```
C#
public static decimal operator + (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to add.

d2 Decimal

The second value to add.

Returns

Decimal

The result of adding `d1` and `d2`.

Implements

Addition(TSelf, TOther)

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Addition method defines the operation of the addition operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 120.07m;
      Decimal number2 = 163.19m;
      Decimal number3 = number1 + number2;
      Console.WriteLine("{0} + {1} = {2}",
                        number1, number2, number3);
   }
}
// The example displays the following output:
//        120.07 + 163.19 = 283.26
```

If the language you're using doesn't support custom operators, call the Add method instead.

The equivalent method for this operator is Decimal.Add(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(Decimal, Decimal)

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Decrement(Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements the Decimal operand by one.

C#

```
public static decimal operator -- (decimal d);
```

Parameters

d Decimal

The value to decrement.

Returns

Decimal

The value of `d` decremented by 1.

Implements

Decrement(TSelf)

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Decrement method defines the operation of the decrement operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number = 1079.8m;
      Console.WriteLine("Original value:    {0:N}", number);
      Console.WriteLine("Decremented value: {0:N}", --number);
   }
}
// The example displays the following output:
//       Original value:    1,079.80
//       Decremented value: 1,078.80
```

Some languages (such as Visual Basic) that lack an increment operator can call the Decrement method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number As Decimal = 1079.8d
      Console.WriteLine("Original value:    {0:N}", number)
      Console.WriteLine("Decremented value: {0:N}",
Decimal.op_Decrement(number))
   End Sub
End Module
' The example displays the following output:
'       Original value:    1,079.80
'       Decremented value: 1,078.80
```

If your language does not support custom operators, call the Subtract method instead, as the following example shows.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number = 1079.8m;
      Console.WriteLine("Original value:    {0:N}", number);
      Console.WriteLine("Decremented value: {0:N}", Decimal.Subtract(number,
1));
   }
}
// The example displays the following output:
```

```
//       Original value:    1,079.80
//       Decremented value: 1,078.80
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Division(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two specified Decimal values.

```
C#
public static decimal operator / (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The dividend.

d2 Decimal

The divisor.

Returns

Decimal

The result of dividing `d1` by `d2`.

Implements

Division(TSelf, TOther)

Exceptions

DivideByZeroException

`d2` is zero.

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Division method defines the operation of the division operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16.8m;
      Decimal number2 = 4.1m;
      Decimal number3 = number1 / number2;
      Console.WriteLine("{0:N2} / {1:N2} = {2:N2}",
                        number1, number2, number3);
   }
}
// The example displays the following output:
//        16.80 / 4.10 = 4.10
```

If the language you're using doesn't support custom operators, call the Divide method instead.

The equivalent method for this operator is Decimal.Divide(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Divide(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Equality(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two Decimal values are equal.

```
C#
public static bool operator == (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` and `d2` are equal; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Remarks

The Equality method defines the operation of the equality operator for Decimal values. It enables code such as the following:

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0695m;
      Decimal number2 = 16354.0699m;
      Console.WriteLine("{0} = {1}: {2}", number1,
                        number2, number1 == number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} = {1}: {2}", number1,
                        number2, number1 == number2);
   }
}
// The example displays the following output:
//       16354.0695 = 16354.0699: False
//       16354.07 = 16354.07: True
```

If the language you're using doesn't support custom operators, call the Equals method instead.

The equivalent method for this operator is Decimal.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- Inequality(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an explicit conversion between a Decimal object and another type.

Overloads

Expand table

Explicit(Single to Decimal)	Defines an explicit conversion of a single-precision floating-point number to a Decimal.
Explicit(Double to Decimal)	Defines an explicit conversion of a double-precision floating-point number to a Decimal.
Explicit(Decimal to UInt64)	Defines an explicit conversion of a Decimal to a 64-bit unsigned integer. This API is not CLS-compliant.
Explicit(Decimal to UInt32)	Defines an explicit conversion of a Decimal to a 32-bit unsigned integer. This API is not CLS-compliant.
Explicit(Decimal to UInt16)	Defines an explicit conversion of a Decimal to a 16-bit unsigned integer. This API is not CLS-compliant.
Explicit(Decimal to SByte)	Defines an explicit conversion of a Decimal to an 8-bit signed integer. This API is not CLS-compliant.
Explicit(Decimal to Single)	Defines an explicit conversion of a Decimal to a single-precision floating-point number.
Explicit(Decimal to Int32)	Defines an explicit conversion of a Decimal to a 32-bit signed integer.

Explicit(Decimal to Int16)	Defines an explicit conversion of a Decimal to a 16-bit signed integer.
Explicit(Decimal to Double)	Defines an explicit conversion of a Decimal to a double-precision floating-point number.
Explicit(Decimal to Char)	Defines an explicit conversion of a Decimal to a Unicode character.
Explicit(Decimal to Byte)	Defines an explicit conversion of a Decimal to an 8-bit unsigned integer.
Explicit(Decimal to Int64)	Defines an explicit conversion of a Decimal to a 64-bit signed integer.

Explicit(Single to Decimal)

Defines an explicit conversion of a single-precision floating-point number to a Decimal.

C#

```
public static explicit operator decimal (float value);
```

Parameters

`value` Single

The single-precision floating-point number to convert.

Returns

Decimal

The converted single-precision floating point number.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

-or-

`value` is NaN, PositiveInfinity, or NegativeInfinity.

Examples

The following example converts Single values to Decimal numbers by using the Single to Decimal conversion operator. This conversion requires the op_Explicit operator in Visual Basic.

C#

```
// Example of the explicit conversion from float to decimal.
using System;

class DecimalFromSingleDemo
{
    const string formatter = "{0,16:E7}{1,33}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the float argument; catch exceptions that are thrown.
    public static void DecimalFromSingle( float argument )
    {
        object decValue;

        // Convert the float argument to a decimal value.
        try
        {
            decValue = (decimal)argument;
        }
        catch( Exception ex )
        {
            decValue = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument, decValue );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversion from float " +
            "to decimal \ngenerates the following output.\n" );
        Console.WriteLine( formatter, "float argument",
            "decimal value" );
        Console.WriteLine( formatter, "--------------",
            "-------------" );

        // Convert float values and display the results.
        DecimalFromSingle( 1.2345E-30F );
```

```
        DecimalFromSingle( 1.2345E-26F );
        DecimalFromSingle( 1.23456E-22F );
        DecimalFromSingle( 1.23456E-12F );
        DecimalFromSingle( 1.234567F );
        DecimalFromSingle( 1.234567E+12F );
        DecimalFromSingle( 1.2345678E+28F );
        DecimalFromSingle( 1.2345678E+30F );
    }
}

/*
This example of the explicit conversion from float to decimal
generates the following output.

  float argument                    decimal value
  --------------                    -------------
  1.2345000E-030                                0
  1.2345000E-026   0.0000000000000000000000000123
  1.2345600E-022    0.0000000000000000000000123456
  1.2345600E-012              0.00000000000123456
  1.2345671E+000                         1.234567
  1.2345670E+012                    1234567000000
  1.2345678E+028    1234568000000000000000000000
  1.2345678E+030                OverflowException
*/
```

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Double to Decimal)

Defines an explicit conversion of a double-precision floating-point number to a Decimal.

```
C#

public static explicit operator decimal (double value);
```

Parameters

`value` Double

The double-precision floating-point number to convert.

Returns

Decimal

The converted double-precision floating point number.

Exceptions

OverflowException

`value` is greater than Decimal.MaxValue or less than Decimal.MinValue.

-or-

`value` is NaN, PositiveInfinity, or NegativeInfinity.

Examples

The following example converts Double values to Decimal numbers by using the Double to Decimal conversion operator. This conversion requires the op_Explicit operator in Visual Basic.

```
C#

// Example of the explicit conversion from double to decimal.
using System;

class DecimalFromDoubleDemo
{
    const string formatter = "{0,25:E16}{1,33}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
```

```
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the double argument; catch exceptions that are thrown.
    public static void DecimalFromDouble( double argument )
    {
        object decValue;

        // Convert the double argument to a decimal value.
        try
        {
            decValue = (decimal)argument;
        }
        catch( Exception ex )
        {
            decValue = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument, decValue );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversion from double " +
            "to decimal \ngenerates the following output.\n" );
        Console.WriteLine( formatter, "double argument",
            "decimal value" );
        Console.WriteLine( formatter, "---------------",
            "-------------" );

        // Convert double values and display the results.
        DecimalFromDouble( 1.234567890123E-30 );
        DecimalFromDouble( 1.2345678901234E-25 );
        DecimalFromDouble( 1.23456789012345E-20 );
        DecimalFromDouble( 1.234567890123456E-10 );
        DecimalFromDouble( 1.2345678901234567 );
        DecimalFromDouble( 1.23456789012345678E+12 );
        DecimalFromDouble( 1.234567890123456789E+28 );
        DecimalFromDouble( 1.234567890123456789E+30 );
    }
}

/*
This example of the explicit conversion from double to decimal
generates the following output.

          double argument                    decimal value
          ---------------                    -------------
  1.2345678901230000E-030                                 0
  1.2345678901233999E-025   0.0000000000000000000000001235
  1.2345678901234499E-020   0.0000000000000000000123456789
  1.2345678901234560E-010       0.0000000001234567890123 46
```

```
  1.2345678901234567E+000                 1.23456789012346
  1.2345678901234568E+012                 1234567890123.46
  1.2345678901234568E+028    1234567890123460000000000000
  1.2345678901234569E+030                OverflowException
*/
```

See also

- Double

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to UInt64)

> Important
>
> This API is not CLS-compliant.

Defines an explicit conversion of a Decimal to a 64-bit unsigned integer.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static explicit operator ulong (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

UInt64

A 64-bit unsigned integer that represents the converted Decimal.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is negative or greater than UInt64.MaxValue.

Examples

The following example converts Decimal numbers to UInt64 values by using the explicit Decimal to UInt64 conversion operator.

C#

```
// Example of the explicit conversions from decimal to long and
// decimal to ulong.
using System;

class DecimalToU_Int64Demo
{
    const string formatter = "{0,25}{1,22}{2,22}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int64( decimal argument )
    {
        object Int64Value;
        object UInt64Value;

        // Convert the argument to a long value.
        try
        {
```

```
            Int64Value = (long)argument;
        }
        catch( Exception ex )
        {
            Int64Value = GetExceptionType( ex );
        }

        // Convert the argument to a ulong value.
        try
        {
            UInt64Value = (ulong)argument;
        }
        catch( Exception ex )
        {
            UInt64Value = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument,
            Int64Value, UInt64Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversions from decimal " +
            "to long \nand decimal to ulong generates the following " +
            "output. It displays \nseveral converted decimal " +
            "values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "long/exception", "ulong/exception" );
        Console.WriteLine( formatter, "----------------",
            "--------------", "---------------" );

        // Convert decimal values and display the results.
        DecimalToU_Int64( 123M );
        DecimalToU_Int64( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int64( 123.999M );
        DecimalToU_Int64( 18446744073709551615.999M );
        DecimalToU_Int64( 18446744073709551616M );
        DecimalToU_Int64( 9223372036854775807.999M );
        DecimalToU_Int64( 9223372036854775808M );
        DecimalToU_Int64( - 0.999M );
        DecimalToU_Int64( - 1M );
        DecimalToU_Int64( - 9223372036854775808.999M );
        DecimalToU_Int64( - 9223372036854775809M );
    }
}

/*
This example of the explicit conversions from decimal to long
and decimal to ulong generates the following output. It displays
several converted decimal values.

         decimal argument        long/exception       ulong/exception
         ----------------        --------------       ---------------
```

```
                        123                    123                    123
                    123.000                    123                    123
                    123.999                    123                    123
  18446744073709551615.999     OverflowException  18446744073709551615
      18446744073709551616     OverflowException     OverflowException
   9223372036854775807.999   9223372036854775807   9223372036854775807
       9223372036854775808     OverflowException   9223372036854775808
                     -0.999                      0                      0
                         -1                     -1     OverflowException
  -9223372036854775808.999  -9223372036854775808     OverflowException
      -9223372036854775809     OverflowException     OverflowException
 */
```

Remarks

This operator supports the explicit conversion of a Decimal to a UInt64. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a UInt64 value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToUInt64(Decimal) or the Convert.ToUInt64(Decimal) method.

See also

- UInt64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to UInt32)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.ToInt64(Decimal)**

Defines an explicit conversion of a Decimal to a 32-bit unsigned integer.

This API is not CLS-compliant.

```
C#

[System.CLSCompliant(false)]
public static explicit operator uint (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

UInt32

A 32-bit unsigned integer that represents the converted Decimal.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example converts Decimal numbers to UInt32 values by using the explicit Decimal to UInt32 conversion operator.

```
C#

// Example of the explicit conversions from decimal to int and
// decimal to uint.
using System;
```

```
class DecimalToU_Int32Demo
{
    const string formatter = "{0,17}{1,19}{2,19}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int32( decimal argument )
    {
        object Int32Value;
        object UInt32Value;

        // Convert the argument to an int value.
        try
        {
            Int32Value = (int)argument;
        }
        catch( Exception ex )
        {
            Int32Value = GetExceptionType( ex );
        }

        // Convert the argument to a uint value.
        try
        {
            UInt32Value = (uint)argument;
        }
        catch( Exception ex )
        {
            UInt32Value = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument,
            Int32Value, UInt32Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversions from decimal " +
            "to int \nand decimal to uint generates the following " +
            "output. It displays \nseveral converted decimal " +
            "values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "int/exception", "uint/exception" );
        Console.WriteLine( formatter, "----------------",
            "-------------", "--------------" );
```

```
        // Convert decimal values and display the results.
        DecimalToU_Int32( 123M );
        DecimalToU_Int32( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int32( 123.999M );
        DecimalToU_Int32( 4294967295.999M );
        DecimalToU_Int32( 4294967296M );
        DecimalToU_Int32( 2147483647.999M );
        DecimalToU_Int32( 2147483648M );
        DecimalToU_Int32( - 0.999M );
        DecimalToU_Int32( - 1M );
        DecimalToU_Int32( - 2147483648.999M );
        DecimalToU_Int32( - 2147483649M );
    }
}

/*
This example of the explicit conversions from decimal to int
and decimal to uint generates the following output. It displays
several converted decimal values.

 decimal argument      int/exception     uint/exception
 ----------------      -------------     --------------
              123                123                123
          123.000                123                123
          123.999                123                123
   4294967295.999  OverflowException         4294967295
       4294967296  OverflowException  OverflowException
   2147483647.999         2147483647         2147483647
       2147483648  OverflowException         2147483648
           -0.999                  0                  0
               -1                 -1  OverflowException
  -2147483648.999        -2147483648  OverflowException
      -2147483649  OverflowException  OverflowException
*/
```

Remarks

This operator supports the explicit conversion of a Decimal to a UInt32. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a UInt32 value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToUInt32(Decimal) or the Convert.ToUInt32(Decimal) method.

See also

- UInt32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to UInt16)

> Important
>
> This API is not CLS-compliant.

Defines an explicit conversion of a Decimal to a 16-bit unsigned integer.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static explicit operator ushort (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

UInt16

A 16-bit unsigned integer that represents the converted Decimal.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example converts Decimal numbers to UInt16 values by using the explicit Decimal to UInt16 conversion operator.

C#

```
// Example of the explicit conversions from decimal to short and
// decimal to ushort.
using System;

class DecimalToU_Int16Demo
{
    const string formatter = "{0,16}{1,19}{2,19}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int16( decimal argument )
    {
        object Int16Value;
        object UInt16Value;

        // Convert the argument to a short value.
        try
        {
            Int16Value = (short)argument;
        }
        catch( Exception ex )
        {
            Int16Value = GetExceptionType( ex );
        }

        // Convert the argument to a ushort value.
        try
        {
            UInt16Value = (ushort)argument;
        }
        catch( Exception ex )
        {
            UInt16Value = GetExceptionType( ex );
```

```
        }

        Console.WriteLine( formatter, argument,
            Int16Value, UInt16Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversions from decimal " +
            "to short \nand decimal to ushort generates the " +
            "following output. It displays \nseveral converted " +
            "decimal values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "short/exception", "ushort/exception" );
        Console.WriteLine( formatter, "----------------",
            "---------------", "----------------" );

        // Convert decimal values and display the results.
        DecimalToU_Int16( 123M );
        DecimalToU_Int16( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int16( 123.999M );
        DecimalToU_Int16( 65535.999M );
        DecimalToU_Int16( 65536M );
        DecimalToU_Int16( 32767.999M );
        DecimalToU_Int16( 32768M );
        DecimalToU_Int16( - 0.999M );
        DecimalToU_Int16( - 1M );
        DecimalToU_Int16( - 32768.999M );
        DecimalToU_Int16( - 32769M );
    }
}

/*
This example of the explicit conversions from decimal to short
and decimal to ushort generates the following output. It displays
several converted decimal values.

decimal argument    short/exception   ushort/exception
----------------    ---------------   ----------------
             123                123                123
         123.000                123                123
         123.999                123                123
       65535.999  OverflowException              65535
           65536  OverflowException  OverflowException
       32767.999              32767              32767
           32768  OverflowException              32768
          -0.999                  0                  0
              -1                 -1  OverflowException
      -32768.999             -32768  OverflowException
          -32769  OverflowException  OverflowException
*/
```

Remarks

This operator supports the explicit conversion of a Decimal to a UInt16. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a UInt16 value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToUInt16(Decimal) or the Convert.ToUInt16(Decimal) method.

See also

- UInt16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to SByte)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.ToInt16(Decimal)**

Defines an explicit conversion of a Decimal to an 8-bit signed integer.

This API is not CLS-compliant.

```
C#
```

```
[System.CLSCompliant(false)]
public static explicit operator sbyte (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

SByte

An 8-bit signed integer that represents the converted Decimal.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` is less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example converts Decimal numbers to SByte values by using the explicit Decimal to SByte conversion operator.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of decimal values.
       decimal[] values = { 78m, new Decimal(78000, 0, 0, false, 3),
                            78.999m, 255.999m, 256m, 127.999m,
                            128m, -0.999m, -1m, -128.999m, -129m };
       foreach (var value in values) {
          try {
              SByte byteValue = (SByte) value;
              Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                                value.GetType().Name, byteValue,
                                byteValue.GetType().Name);
           }
```

```
            catch (OverflowException) {
               Console.WriteLine("OverflowException: Cannot convert {0}",
                                 value);
            }
        }
    }
}
// The example displays the following output:
//       78 (Decimal) --> 78 (SByte)
//       78.000 (Decimal) --> 78 (SByte)
//       78.999 (Decimal) --> 78 (SByte)
//       OverflowException: Cannot convert 255.999
//       OverflowException: Cannot convert 256
//       127.999 (Decimal) --> 127 (SByte)
//       OverflowException: Cannot convert 128
//       -0.999 (Decimal) --> 0 (SByte)
//       -1 (Decimal) --> -1 (SByte)
//       -128.999 (Decimal) --> -128 (SByte)
//       OverflowException: Cannot convert -129
```

Remarks

This operator supports the explicit conversion of a Decimal to a SByte. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a Byte and an SByte value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToSByte(Decimal) method or the Convert.ToSByte(Decimal) method.

See also

- SByte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Single)

Defines an explicit conversion of a Decimal to a single-precision floating-point number.

C#

```
public static explicit operator float (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Single

A single-precision floating-point number that represents the converted Decimal.

Examples

The following example converts Decimal numbers to Single values by using the explicit Decimal to Single conversion operator.

C#

```
using System;

class Example
{
    public static void Main( )
    {
        // Define an array of decimal values.
        decimal[] values = { 0.0000000000000000000000000001M,
                             0.0000000000123456789123456789M,
                             123M, new decimal(123000000, 0, 0, false,
6),
                             123456789.123456789M,
```

```
                                 1234567891234567891234567 89M,
                                 decimal.MinValue, decimal.MaxValue };
            // Convert each value to a double.
            foreach (var value in values) {
                float dblValue = (float) value;
                Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                                  value.GetType().Name, dblValue,
                                  dblValue.GetType().Name);
            }
        }
    }
// The example displays the following output:
//    0.0000000000000000000000000001 (Decimal) --> 1E-28 (Single)
//    0.0000000000123456789123456789 (Decimal) --> 1.234568E-11 (Single)
//    123 (Decimal) --> 123 (Single)
//    123.000000 (Decimal) --> 123 (Single)
//    123456789.123456789 (Decimal) --> 1.234568E+08 (Single)
//    1234567891234567891234567 89 (Decimal) --> 1.234568E+26 (Single)
//    -7922816251426433759354395033 5 (Decimal) --> -7.922816E+28 (Single)
//    7922816251426433759354395033 5 (Decimal) --> 7.922816E+28 (Single)
```

Remarks

This operation can result in a loss of precision, because a single-precision floating-point number has fewer significant digits than a Decimal.

This operator supports the explicit conversion of a Decimal to a Single. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a Single value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToSingle(Decimal) or the Convert.ToSingle(Decimal) method.

See also

- Single

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Int32)

Defines an explicit conversion of a Decimal to a 32-bit signed integer.

```
C#

public static explicit operator int (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Int32

A 32-bit signed integer that represents the converted Decimal.

Exceptions

OverflowException

`value` is less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example converts Decimal numbers to Int32 values by using the explicit Decimal to Int32 conversion operator.

```
C#

// Example of the explicit conversions from decimal to int and
// decimal to uint.
using System;
```

```
class DecimalToU_Int32Demo
{
    const string formatter = "{0,17}{1,19}{2,19}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int32( decimal argument )
    {
        object Int32Value;
        object UInt32Value;

        // Convert the argument to an int value.
        try
        {
            Int32Value = (int)argument;
        }
        catch( Exception ex )
        {
            Int32Value = GetExceptionType( ex );
        }

        // Convert the argument to a uint value.
        try
        {
            UInt32Value = (uint)argument;
        }
        catch( Exception ex )
        {
            UInt32Value = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument,
            Int32Value, UInt32Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversions from decimal " +
            "to int \nand decimal to uint generates the following " +
            "output. It displays \nseveral converted decimal " +
            "values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "int/exception", "uint/exception" );
        Console.WriteLine( formatter, "----------------",
            "-------------", "--------------" );
```

```
        // Convert decimal values and display the results.
        DecimalToU_Int32( 123M );
        DecimalToU_Int32( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int32( 123.999M );
        DecimalToU_Int32( 4294967295.999M );
        DecimalToU_Int32( 4294967296M );
        DecimalToU_Int32( 2147483647.999M );
        DecimalToU_Int32( 2147483648M );
        DecimalToU_Int32( - 0.999M );
        DecimalToU_Int32( - 1M );
        DecimalToU_Int32( - 2147483648.999M );
        DecimalToU_Int32( - 2147483649M );
    }
}

/*
This example of the explicit conversions from decimal to int
and decimal to uint generates the following output. It displays
several converted decimal values.

 decimal argument      int/exception     uint/exception
 ----------------      -------------     --------------
              123                123                123
          123.000                123                123
          123.999                123                123
   4294967295.999  OverflowException         4294967295
       4294967296  OverflowException  OverflowException
   2147483647.999         2147483647         2147483647
       2147483648  OverflowException         2147483648
           -0.999                  0                  0
               -1                 -1  OverflowException
  -2147483648.999        -2147483648  OverflowException
      -2147483649  OverflowException  OverflowException
*/
```

Remarks

This operator supports the explicit conversion of a Decimal to an Int32. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to an Int32 value by using C#, C++, and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToInt32(Decimal) or the Convert.ToInt32(Decimal) method.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Int16)

Defines an explicit conversion of a Decimal to a 16-bit signed integer.

C#

```
public static explicit operator short (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Int16

A 16-bit signed integer that represents the converted Decimal.

Exceptions

OverflowException

`value` is less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example converts Decimal numbers to Int16 values by using the explicit Decimal to Int16 conversion operator.

C#

```
// Example of the explicit conversions from decimal to short and
// decimal to ushort.
using System;

class DecimalToU_Int16Demo
{
    const string formatter = "{0,16}{1,19}{2,19}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int16( decimal argument )
    {
        object Int16Value;
        object UInt16Value;

        // Convert the argument to a short value.
        try
        {
            Int16Value = (short)argument;
        }
        catch( Exception ex )
        {
            Int16Value = GetExceptionType( ex );
        }

        // Convert the argument to a ushort value.
        try
        {
            UInt16Value = (ushort)argument;
        }
        catch( Exception ex )
        {
            UInt16Value = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument,
            Int16Value, UInt16Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
```

```
            "This example of the explicit conversions from decimal " +
            "to short \nand decimal to ushort generates the " +
            "following output. It displays \nseveral converted " +
            "decimal values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "short/exception", "ushort/exception" );
        Console.WriteLine( formatter, "----------------",
            "---------------", "----------------" );

        // Convert decimal values and display the results.
        DecimalToU_Int16( 123M );
        DecimalToU_Int16( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int16( 123.999M );
        DecimalToU_Int16( 65535.999M );
        DecimalToU_Int16( 65536M );
        DecimalToU_Int16( 32767.999M );
        DecimalToU_Int16( 32768M );
        DecimalToU_Int16( - 0.999M );
        DecimalToU_Int16( - 1M );
        DecimalToU_Int16( - 32768.999M );
        DecimalToU_Int16( - 32769M );
    }
}

/*
This example of the explicit conversions from decimal to short
and decimal to ushort generates the following output. It displays
several converted decimal values.

decimal argument    short/exception   ushort/exception
----------------    ---------------   ----------------
             123                123                123
         123.000                123                123
         123.999                123                123
       65535.999  OverflowException              65535
           65536  OverflowException  OverflowException
       32767.999              32767              32767
           32768  OverflowException              32768
          -0.999                  0                  0
              -1                 -1  OverflowException
      -32768.999             -32768  OverflowException
          -32769  OverflowException  OverflowException
*/
```

Remarks

This operator supports the explicit conversion of a Decimal to an Int16. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to an Int16 value by using C#, Visual Basic, and C++. To perform a conversion

that is independent of language, you can call the Decimal.ToInt16(Decimal) or the Convert.ToInt16(Decimal) method.

See also

- Int16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Double)

Defines an explicit conversion of a Decimal to a double-precision floating-point number.

C#

```
public static explicit operator double (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Double

A double-precision floating-point number that represents the converted Decimal.

Examples

The following example converts Decimal numbers to Double values by using the explicit Decimal to Double conversion operator.

C#

```
using System;

class Example
{
    public static void Main( )
    {
        // Define an array of decimal values.
        decimal[] values = { 0.0000000000000000000000000001M,
                             0.0000000000123456789123456789M,
                             123M, new decimal(123000000, 0, 0, false,
6),
                             123456789.123456789M,
                             1234567891234567891234567 89M,
                             decimal.MinValue, decimal.MaxValue };
        // Convert each value to a double.
        foreach (var value in values) {
            double dblValue = (double) value;
            Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                              value.GetType().Name, dblValue,
                              dblValue.GetType().Name);
       }
    }
}
// The example displays the following output:
//    0.0000000000000000000000000001 (Decimal) --> 1E-28 (Double)
//    0.0000000000123456789123456789 (Decimal) --> 1.23456789123457E-11
(Double)
//    123 (Decimal) --> 123 (Double)
//    123.000000 (Decimal) --> 123 (Double)
//    123456789.123456789 (Decimal) --> 123456789.123457 (Double)
//    1234567891234567891234567 89 (Decimal) --> 1.23456789123457E+26
(Double)
//    -7922816251426433759354395033 5 (Decimal) --> -7.92281625142643E+28
(Double)
//    7922816251426433759354395033 5 (Decimal) --> 7.92281625142643E+28
(Double)
```

Remarks

This operation can result in a loss of precision, because a double-precision floating-point number has fewer significant digits than a Decimal.

This operator supports the explicit conversion of a Decimal to a Double. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a Double value by using C#, C++, and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToDouble(Decimal) or the Convert.ToDouble(Decimal) method.

See also

- Double

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Char)

Defines an explicit conversion of a Decimal to a Unicode character.

C#

```
public static explicit operator char (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Char

A Unicode character that represents the converted Decimal.

Exceptions

OverflowException

`value` is less than Char.MinValue or greater than Char.MaxValue.

Examples

The following example converts Decimal numbers to Char values (Unicode characters) by using the explicit Decimal to Char conversion operator.

C#

```
using System;

class Example
{
   public static void Main( )
   {
      // Define an array of decimal values.
      decimal[] values = { 3.33m, 55.5m, 77.7m, 123m, 123.999m, 170m,
                           188.88m, 222m, 244m, 8217m, 8250m, 65536m,
                           -1m };
      // Convert each value to a Char.
      foreach (decimal value in values) {
         try {
            char charValue = (char) value;
            Console.WriteLine("{0} --> {1} ({2:X4})", value,
                              charValue, (ushort) charValue);
         }
         catch (OverflowException) {
            Console.WriteLine("OverflowException: Cannot convert {0}",
                              value);
         }
      }
   }
}
// The example displays the following output:
//       3.33 --> ? (0003)
//       55.5 --> 7 (0037)
//       77.7 --> M (004D)
//       123 --> { (007B)
//       123.999 --> { (007B)
//       170 --> ª (00AA)
//       188.88 --> ¼ (00BC)
//       222 --> _ (00DE)
//       244 --> ô (00F4)
```

```
//       8217 --> ' (2019)
//       8250 --> > (203A)
//       OverflowException: Cannot convert 65536
//       OverflowException: Cannot convert -1
```

Remarks

This operator supports the explicit conversion of a Decimal to a Char. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results, or might not support the conversion at all. For example, the Visual Basic and C++ compilers do not support an explicit Decimal to a Char conversion.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Byte)

Defines an explicit conversion of a Decimal to an 8-bit unsigned integer.

C#

```
public static explicit operator byte (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Byte

An 8-bit unsigned integer that represents the converted Decimal.

Exceptions

OverflowException

`value` is less than Byte.MinValue or greater than Byte.MaxValue.

Examples

The following example converts Decimal numbers to Byte values by using the explicit Decimal to Byte conversion operator.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of decimal values.
       decimal[] values = { 78m, new Decimal(78000, 0, 0, false, 3),
                            78.999m, 255.999m, 256m, 127.999m,
                            128m, -0.999m, -1m, -128.999m, -129m };
       foreach (var value in values) {
          try {
              Byte byteValue = (Byte) value;
              Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                                value.GetType().Name, byteValue,
                                byteValue.GetType().Name);
           }
           catch (OverflowException) {
              Console.WriteLine("OverflowException: Cannot convert {0}",
                                value);
           }
       }
    }
}
// The example displays the following output:
//       78 (Decimal) --> 78 (Byte)
//       78.000 (Decimal) --> 78 (Byte)
//       78.999 (Decimal) --> 78 (Byte)
```

```
//       255.999 (Decimal) --> 255 (Byte)
//       OverflowException: Cannot convert 256
//       127.999 (Decimal) --> 127 (Byte)
//       128 (Decimal) --> 128 (Byte)
//       -0.999 (Decimal) --> 0 (Byte)
//       OverflowException: Cannot convert -1
//       OverflowException: Cannot convert -128.999
//       OverflowException: Cannot convert -129
```

Remarks

This operator supports the explicit conversion of a Decimal to a Byte. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to a Byte value by using C#, C++, and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToByte(Decimal) or the Convert.ToByte(Decimal) method.

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Decimal to Int64)

Defines an explicit conversion of a Decimal to a 64-bit signed integer.

```C#
public static explicit operator long (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Int64

A 64-bit signed integer that represents the converted Decimal.

Exceptions

OverflowException

`value` is less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example converts Decimal numbers to Int64 values by using the explicit Decimal to Int64 conversion operator.

```C#
// Example of the explicit conversions from decimal to long and
// decimal to ulong.
using System;

class DecimalToU_Int64Demo
{
    const string formatter = "{0,25}{1,22}{2,22}";

    // Get the exception type name; remove the namespace prefix.
    public static string GetExceptionType( Exception ex )
    {
        string exceptionType = ex.GetType( ).ToString( );
        return exceptionType.Substring(
            exceptionType.LastIndexOf( '.' ) + 1 );
    }

    // Convert the decimal argument; catch exceptions that are thrown.
    public static void DecimalToU_Int64( decimal argument )
    {
        object Int64Value;
```

```
        object UInt64Value;

        // Convert the argument to a long value.
        try
        {
            Int64Value = (long)argument;
        }
        catch( Exception ex )
        {
            Int64Value = GetExceptionType( ex );
        }

        // Convert the argument to a ulong value.
        try
        {
            UInt64Value = (ulong)argument;
        }
        catch( Exception ex )
        {
            UInt64Value = GetExceptionType( ex );
        }

        Console.WriteLine( formatter, argument,
            Int64Value, UInt64Value );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the explicit conversions from decimal " +
            "to long \nand decimal to ulong generates the following " +
            "output. It displays \nseveral converted decimal " +
            "values.\n" );
        Console.WriteLine( formatter, "decimal argument",
            "long/exception", "ulong/exception" );
        Console.WriteLine( formatter, "----------------",
            "--------------", "---------------" );

        // Convert decimal values and display the results.
        DecimalToU_Int64( 123M );
        DecimalToU_Int64( new decimal( 123000, 0, 0, false, 3 ) );
        DecimalToU_Int64( 123.999M );
        DecimalToU_Int64( 18446744073709551615.999M );
        DecimalToU_Int64( 18446744073709551616M );
        DecimalToU_Int64( 9223372036854775807.999M );
        DecimalToU_Int64( 9223372036854775808M );
        DecimalToU_Int64( - 0.999M );
        DecimalToU_Int64( - 1M );
        DecimalToU_Int64( - 9223372036854775808.999M );
        DecimalToU_Int64( - 9223372036854775809M );
    }
}

/*
This example of the explicit conversions from decimal to long
```

```
and decimal to ulong generates the following output. It displays
several converted decimal values.

         decimal argument        long/exception       ulong/exception
         ----------------        --------------       ---------------
                      123                   123                   123
                  123.000                   123                   123
                  123.999                   123                   123
 18446744073709551615.999     OverflowException  18446744073709551615
     18446744073709551616     OverflowException     OverflowException
  9223372036854775807.999   9223372036854775807   9223372036854775807
      9223372036854775808     OverflowException   9223372036854775808
                   -0.999                     0                     0
                       -1                    -1     OverflowException
 -9223372036854775808.999  -9223372036854775808     OverflowException
     -9223372036854775809     OverflowException     OverflowException
*/
```

Remarks

This operator supports the explicit conversion of a Decimal to an Int64. The syntax for such explicit conversions is language-dependent, and individual language compilers can provide different implementations and return different results. The example illustrates the different return values when you explicitly convert a Decimal value to an Int64 value by using C# and Visual Basic. To perform a conversion that is independent of language, you can call the Decimal.ToInt64(Decimal) or the Convert.ToInt64(Decimal) method.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.GreaterThan(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether a specified Decimal is greater than another specified Decimal.

C#

```
public static bool operator > (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` is greater than `d2`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Remarks

The GreaterThan method defines the operation of the greater than operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0699m;
      Decimal number2 = 16354.0695m;
      Console.WriteLine("{0} > {1}: {2}", number1,
                        number2, number1 > number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} > {1}: {2}", number1,
                        number2, number1 > number2);
   }
}
// The example displays the following output:
//       16354.0699 > 16354.0695: True
//       16354.07 > 16354.07: False
```

Languages that do not support custom operators can call the Compare method instead. They may also be able to call the GreaterThan method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number1 As Decimal = 16354.0699d
      Dim number2 As Decimal = 16354.0695d
      Console.WriteLine("{0} > {1}: {2}", number1, number2,
                        Decimal.op_GreaterThan(number1, number2))

      number1 = Decimal.Round(number1, 2)
      number2 = Decimal.Round(number2, 2)
      Console.WriteLine("{0} > {1}: {2}", number1, number2,
                        Decimal.op_GreaterThan(number1, number2))
   End Sub
End Module
' The example displays the following output:
'       16354.0699 > 16354.0695: True
'       16354.07 > 16354.07: False
```

The equivalent method for this operator is Decimal.Compare(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.GreaterThanOrEqual(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether a specified Decimal is greater than or equal to another specified Decimal.

C#

```
public static bool operator >= (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` is greater than or equal to `d2`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Remarks

The GreaterThanOrEqual method defines the operation of the greater than or equal operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0699m;
      Decimal number2 = 16354.0695m;
      Console.WriteLine("{0} >= {1}: {2}", number1,
                        number2, number1 >= number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} >= {1}: {2}", number1,
                        number2, number1 >= number2);
   }
}
// The example displays the following output:
//       16354.0699 >= 16354.0695: True
//       16354.07 >= 16354.07: True
```

Languages that do not support custom operators can call the Compare method instead. They may also be able to call the GreaterThanOrEqual method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number1 As Decimal = 16354.0699d
      Dim number2 As Decimal = 16354.0695d
      Console.WriteLine("{0} >= {1}: {2}", number1, number2,
                        Decimal.op_GreaterThanOrEqual(number1, number2))

      number1 = Decimal.Round(number1, 2)
      number2 = Decimal.Round(number2, 2)
      Console.WriteLine("{0} >= {1}: {2}", number1, number2,
                        Decimal.op_GreaterThanOrEqual(number1, number2))
   End Sub
End Module
' The example displays the following output:
'       16354.0699 >= 16354.0695: True
'       16354.07 >= 16354.07: True
```

The equivalent method for this operator is Decimal.Compare(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an implicit conversion of the value of a type to a Decimal value.

Overloads

Expand table

Implicit(UInt16 to Decimal)	Defines an implicit conversion of a 16-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Implicit(UInt64 to Decimal)	Defines an implicit conversion of a 64-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Implicit(UInt32 to Decimal)	Defines an implicit conversion of a 32-bit unsigned integer to a Decimal. This API is not CLS-compliant.
Implicit(Int64 to Decimal)	Defines an implicit conversion of a 64-bit signed integer to a Decimal.
Implicit(SByte to Decimal)	Defines an implicit conversion of an 8-bit signed integer to a Decimal. This API is not CLS-compliant.
Implicit(Int16 to Decimal)	Defines an implicit conversion of a 16-bit signed integer to a Decimal.
Implicit(Char to Decimal)	Defines an implicit conversion of a Unicode character to a Decimal.
Implicit(Byte to Decimal)	Defines an implicit conversion of an 8-bit unsigned integer to a Decimal.

Implicit(Int32 to Decimal)	Defines an implicit conversion of a 32-bit signed integer to a Decimal.

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

Implicit(UInt16 to Decimal)

> Important
>
> This API is not CLS-compliant.

Defines an implicit conversion of a 16-bit unsigned integer to a Decimal.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static implicit operator decimal (ushort value);
```

Parameters

`value` UInt16

The 16-bit unsigned integer to convert.

Returns

Decimal

The converted 16-bit unsigned integer.

Attributes CLSCompliantAttribute

Examples

The following example converts UInt16 values to Decimal numbers.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of 16-bit unsigned integer values.
        ushort[] values = { ushort.MinValue, ushort.MaxValue,
                            0xFFF, 12345, 40000 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//       0 (UInt16) --> 0 (Decimal)
//       65535 (UInt16) --> 65535 (Decimal)
//       4095 (UInt16) --> 4095 (Decimal)
//       12345 (UInt16) --> 12345 (Decimal)
//       40000 (UInt16) --> 40000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- UInt16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(UInt64 to Decimal)

> Important
>
> This API is not CLS-compliant.

Defines an implicit conversion of a 64-bit unsigned integer to a Decimal.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static implicit operator decimal (ulong value);
```

Parameters

`value` UInt64

The 64-bit unsigned integer to convert.

Returns

Decimal

The converted 64-bit unsigned integer.

Attributes CLSCompliantAttribute

Examples

The following example converts UInt64 values to Decimal numbers by using the UInt64 to Decimal conversion operator.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        // Define an array of 64-bit unsigned integer values.
        ulong[] values = { ulong.MinValue, ulong.MaxValue,
                           0xFFFFFFFFFFFF, 123456789123456789,
                           1000000000000000 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//    0 (UInt64) --> 0 (Decimal)
//    18446744073709551615 (UInt64) --> 18446744073709551615 (Decimal)
//    281474976710655 (UInt64) --> 281474976710655 (Decimal)
//    123456789123456789 (UInt64) --> 123456789123456789 (Decimal)
//    1000000000000000 (UInt64) --> 1000000000000000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- UInt64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(UInt32 to Decimal)

> Important
>
> This API is not CLS-compliant.

Defines an implicit conversion of a 32-bit unsigned integer to a Decimal.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static implicit operator decimal (uint value);
```

Parameters

`value` UInt32

The 32-bit unsigned integer to convert.

Returns

Decimal

The converted 32-bit unsigned integer.

Attributes CLSCompliantAttribute

Examples

The following example converts UInt32 values to Decimal numbers.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of 32-bit unsigned integer values.
        uint[] values = { uint.MinValue, uint.MaxValue, 0xFFFFFF,
                          123456789, 4000000000 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//       0 (UInt32) --> 0 (Decimal)
//       4294967295 (UInt32) --> 4294967295 (Decimal)
//       16777215 (UInt32) --> 16777215 (Decimal)
//       123456789 (UInt32) --> 123456789 (Decimal)
//       4000000000 (UInt32) --> 4000000000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- UInt32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(Int64 to Decimal)

Defines an implicit conversion of a 64-bit signed integer to a Decimal.

C#

```
public static implicit operator decimal (long value);
```

Parameters

`value` Int64

The 64-bit signed integer to convert.

Returns

Decimal

The converted 64-bit signed integer.

Examples

The following example converts Int64 values to Decimal numbers.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of 64-bit integer values.
        long[] values = { long.MinValue, long.MaxValue,
```

```
                            0xFFFFFFFFFFFF, 123456789123456789,
                            -1000000000000000 };
         // Convert each value to a Decimal.
         foreach (var value in values) {
            Decimal decValue = value;
            Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                              value.GetType().Name, decValue,
                              decValue.GetType().Name);
         }
      }
   }
   // The example displays the following output:
   //    -9223372036854775808 (Int64) --> -9223372036854775808 (Decimal)
   //    9223372036854775807 (Int64) --> 9223372036854775807 (Decimal)
   //    281474976710655 (Int64) --> 281474976710655 (Decimal)
   //    123456789123456789 (Int64) --> 123456789123456789 (Decimal)
   //    -1000000000000000 (Int64) --> -1000000000000000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(SByte to Decimal)

> Important
>
> This API is not CLS-compliant.

Defines an implicit conversion of an 8-bit signed integer to a Decimal.

This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static implicit operator decimal (sbyte value);
```

Parameters

`value` SByte

The 8-bit signed integer to convert.

Returns

Decimal

The converted 8-bit signed integer.

Attributes CLSCompliantAttribute

Examples

The following example converts SByte values to Decimal numbers.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of 8-bit signed integer values.
        sbyte[] values = { sbyte.MinValue, sbyte.MaxValue,
                           0x3F, 123, -100 };
        // Convert each value to a Decimal.
```

```
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//    -128 (SByte) --> -128 (Decimal)
//    127 (SByte) --> 127 (Decimal)
//    63 (SByte) --> 63 (Decimal)
//    123 (SByte) --> 123 (Decimal)
//    -100 (SByte) --> -100 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- SByte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(Int16 to Decimal)

Defines an implicit conversion of a 16-bit signed integer to a Decimal.

C#

```
public static implicit operator decimal (short value);
```

Parameters

`value` Int16

The 16-bit signed integer to convert.

Returns

Decimal

The converted 16-bit signed integer.

Examples

The following example converts Int16 values to Decimal numbers. This conversion requires the op_Implicit operator in Visual Basic, but not in C# and C++.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of 16-bit integer values.
        short[] values = { short.MinValue, short.MaxValue,
                           0xFFF, 12345, -10000 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//       -32768 (Int16) --> -32768 (Decimal)
//       32767 (Int16) --> 32767 (Decimal)
//       4095 (Int16) --> 4095 (Decimal)
//       12345 (Int16) --> 12345 (Decimal)
//       -10000 (Int16) --> -10000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- Int16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(Char to Decimal)

Defines an implicit conversion of a Unicode character to a Decimal.

C#

```
public static implicit operator decimal (char value);
```

Parameters

`value` Char

The Unicode character to convert.

Returns

Decimal

The converted Unicode character.

Examples

The following example converts Char values (Unicode characters) to Decimal numbers. This conversion requires the op_Implicit operator in Visual Basic, but not in C# and C++.

C#

```
using System;

class Example
{
    public static void Main()
    {
        // Define an array of Char values.
        Char[] values = { '\0', ' ', '*', 'A', 'a', '{', 'Æ' };

        // Convert each Char value to a Decimal.
        foreach (var value in values) {
           decimal decValue = value;
           Console.WriteLine("'{0}' ({1}) --> {2} ({3})", value,
                              value.GetType().Name, decValue,
                              decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//       ' ' (Char) --> 0 (Decimal)
//       ' ' (Char) --> 32 (Decimal)
//       '*' (Char) --> 42 (Decimal)
//       'A' (Char) --> 65 (Decimal)
//       'a' (Char) --> 97 (Decimal)
//       '{' (Char) --> 123 (Decimal)
//       'Æ' (Char) --> 198 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- Char

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Implicit(Byte to Decimal)

Defines an implicit conversion of an 8-bit unsigned integer to a Decimal.

C#

```
public static implicit operator decimal (byte value);
```

Parameters

`value` Byte

The 8-bit unsigned integer to convert.

Returns

Decimal

The converted 8-bit unsigned integer.

Examples

The following example converts Byte values to Decimal numbers.

C#

```
using System;

class Example
```

```
{
    public static void Main()
    {
        // Define an array of byte values.
        byte[] values = { byte.MinValue, byte.MaxValue,
                          0x3F, 123, 200 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
    }
}
// The example displays the following output:
//       0 (Byte) --> 0 (Decimal)
//       255 (Byte) --> 255 (Decimal)
//       63 (Byte) --> 63 (Decimal)
//       123 (Byte) --> 123 (Decimal)
//       200 (Byte) --> 200 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Implicit(Int32 to Decimal)

Defines an implicit conversion of a 32-bit signed integer to a Decimal.

```
C#

public static implicit operator decimal (int value);
```

Parameters

`value` Int32

The 32-bit signed integer to convert.

Returns

Decimal

The converted 32-bit signed integer.

Examples

The following example converts Int32 values to Decimal numbers.

```
C#

using System;

class Example
{
    public static void Main()
    {
        // Define an array of 32-bit integer values.
        int[] values = { int.MinValue, int.MaxValue, 0xFFFFFF,
                         123456789, -1000000000 };
        // Convert each value to a Decimal.
        foreach (var value in values) {
           Decimal decValue = value;
           Console.WriteLine("{0} ({1}) --> {2} ({3})", value,
                             value.GetType().Name, decValue,
                             decValue.GetType().Name);
        }
```

```
    }
}
// The example displays the following output:
//    -2147483648 (Int32) --> -2147483648 (Decimal)
//    2147483647 (Int32) --> 2147483647 (Decimal)
//    16777215 (Int32) --> 16777215 (Decimal)
//    123456789 (Int32) --> 123456789 (Decimal)
//    -1000000000 (Int32) --> -1000000000 (Decimal)
```

Remarks

The overloads of the Implicit method define the types from which the compiler can automatically convert a Decimal value without an explicit casting operator (in C#) or a call to a conversion function (in Visual Basic). They are widening conversions that do not involve data loss and do not throw an OverflowException exception.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Decimal.Increment(Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments the Decimal operand by 1.

C#

```
public static decimal operator ++ (decimal d);
```

Parameters

d Decimal

The value to increment.

Returns

Decimal

The value of `d` incremented by 1.

Implements

Increment(TSelf)

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Increment method defines the operation of the increment operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number = 1079.8m;
      Console.WriteLine("Original value:    {0:N}", number);
      Console.WriteLine("Incremented value: {0:N}", ++number);
   }
}
// The example displays the following output:
//       Original value:    1,079.80
//       Incremented value: 1,080.80
```

Some languages (such as Visual Basic) that lack an increment operator can call the Increment method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number As Decimal = 1079.8d
      Console.WriteLine("Original value:    {0:N}", number)
      Console.WriteLine("Incremented value: {0:N}",
Decimal.op_Increment(number))
   End Sub
End Module
' The example displays the following output:
'       Original value:    1,079.80
'       Incremented value: 1,080.80
```

If your language does not support custom operators, call the Add method instead, as the following example shows.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number = 1079.8m;
      Console.WriteLine("Original value:    {0:N}", number);
      Console.WriteLine("Incremented value: {0:N}", Decimal.Add(number, 1));
   }
}
// The example displays the following output:
```

```
//       Original value:    1,079.80
//       Incremented value: 1,080.80
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Inequality(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two Decimal objects have different values.

```
C#
public static bool operator != (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` and `d2` are not equal; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Remarks

The Inequality method defines the operation of the inequality operator for Decimal values. It enables code such as the following:

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0695m;
      Decimal number2 = 16354.0699m;
      Console.WriteLine("{0} <> {1}: {2}", number1,
                        number2, number1 != number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} <> {1}: {2}", number1,
                        number2, number1 != number2);
   }
}
// The example displays the following output:
//       16354.0695 <> 16354.0699: True
//       16354.07 <> 16354.07: False
```

If the language you're using doesn't support custom operators, you can test for inequality by using one of the following techniques:

- Calling the Compare method, which indicates the relationship between two Decimal values.
- Calling the Equals method and reversing its value.

The equivalent method for this operator is Decimal.Compare(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- Equality(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.LessThan(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether a specified Decimal is less than another specified Decimal.

C#

```
public static bool operator < (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` is less than `d2`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Remarks

The LessThan method defines the operation of the less than operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0699m;
      Decimal number2 = 16354.0695m;
      Console.WriteLine("{0} < {1}: {2}", number1,
                        number2, number1 < number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} < {1}: {2}", number1,
                        number2, number1 < number2);
   }
}
// The example displays the following output:
//       16354.0699 < 16354.0695: False
//       16354.07 < 16354.07: False
```

Languages that do not support custom operators can call the Compare method instead. They may also be able to call the LessThan method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number1 As Decimal = 16354.0699d
      Dim number2 As Decimal = 16354.0695d
      Console.WriteLine("{0} < {1}: {2}", number1, number2,
                        Decimal.op_LessThan(number1, number2))

      number1 = Decimal.Round(number1, 2)
      number2 = Decimal.Round(number2, 2)
      Console.WriteLine("{0} < {1}: {2}", number1, number2,
                        Decimal.op_LessThan(number1, number2))
   End Sub
End Module
' The example displays the following output:
'       16354.0699 < 16354.0695: False
'       16354.07 < 16354.07: False
```

The equivalent method for this operator is Decimal.Compare(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.LessThanOrEqual(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether a specified Decimal is less than or equal to another specified Decimal.

```
C#
public static bool operator <= (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to compare.

d2 Decimal

The second value to compare.

Returns

Boolean

`true` if `d1` is less than or equal to `d2`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Remarks

The LessThanOrEqual method defines the operation of the less than or equal operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16354.0699m;
      Decimal number2 = 16354.0695m;
      Console.WriteLine("{0} <= {1}: {2}", number1,
                        number2, number1 <= number2);

      number1 = Decimal.Round(number1, 2);
      number2 = Decimal.Round(number2, 2);
      Console.WriteLine("{0} <= {1}: {2}", number1,
                        number2, number1 <= number2);
   }
}
// The example displays the following output:
//       16354.0699 <= 16354.0695: False
//       16354.07 <= 16354.07: True
```

Languages that do not support custom operators can call the Compare method instead. They may also be able to call the LessThanOrEqual method directly, as the following example shows.

VB

```
Module Example
   Public Sub Main()
      Dim number1 As Decimal = 16354.0699d
      Dim number2 As Decimal = 16354.0695d
      Console.WriteLine("{0} <= {1}: {2}", number1, number2,
                        Decimal.op_LessThanOrEqual(number1, number2))

      number1 = Decimal.Round(number1, 2)
      number2 = Decimal.Round(number2, 2)
      Console.WriteLine("{0} <= {1}: {2}", number1, number2,
                        Decimal.op_LessThanOrEqual(number1, number2))
   End Sub
End Module
' The example displays the following output:
'       16354.0699 >= 16354.0695: False
'       16354.07 >= 16354.07: True
```

The equivalent method for this operator is Decimal.Compare(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Modulus(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the remainder resulting from dividing two specified Decimal values.

```
C#
public static decimal operator % (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The dividend.

d2 Decimal

The divisor.

Returns

Decimal

The remainder resulting from dividing `d1` by `d2`.

Implements

Modulus(TSelf, TOther)

Exceptions

DivideByZeroException

`d2` is `zero`.

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Modulus operator defines the remainder operation that returns the remainder resulting from dividing two specified Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16.8m;
      Decimal number2 = 4.1m;
      Decimal number3 = number1 % number2;
      Console.WriteLine("{0:N2} % {1:N2} = {2:N2}",
                        number1, number2, number3);
   }
}
// The example displays the following output:
//        16.80 % 4.10 = 0.40
```

The sign of the value returned by the remainder operation depends on the sign of dividend. If dividend is positive, the remainder operation returns a positive result; if it is negative, the remainder operation returns a negative result.

The equivalent method for this operator is Decimal.Remainder(Decimal, Decimal). If the language you're using doesn't support custom operators, call the Remainder method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Remainder(Decimal, Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.Multiply(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two specified Decimal values.

C#

```
public static decimal operator * (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The first value to multiply.

d2 Decimal

The second value to multiply.

Returns

Decimal

The result of multiplying `d1` by `d2`.

Implements

Multiply(TSelf, TOther)

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Multiply method defines the operation of the multiplication operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 16.8m;
      Decimal number2 = 4.1m;
      Decimal number3 = number1 * number2;
      Console.WriteLine("{0:N2} x {1:N2} = {2:N2}",
                        number1, number2, number3);
   }
}
// The example displays the following output:
//        16.80 x 4.10 = 68.88
```

If the language you're using doesn't support custom operators, call the Multiply method instead.

The equivalent method for this operator is Decimal.Multiply(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Multiply(Decimal, Decimal)

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.Subtraction(Decimal, Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two specified Decimal values.

```
C#
public static decimal operator - (decimal d1, decimal d2);
```

Parameters

d1 Decimal

The minuend.

d2 Decimal

The subtrahend.

Returns

Decimal

The result of subtracting `d2` from `d1`.

Implements

Subtraction(TSelf, TOther)

Exceptions

OverflowException

The return value is less than Decimal.MinValue or greater than Decimal.MaxValue.

Remarks

The Subtraction method defines the operation of the subtraction operator for Decimal values. It enables code such as the following:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Decimal number1 = 120.07m;
      Decimal number2 = 163.19m;
      Decimal number3 = number1 - number2;
      Console.WriteLine("{0} - {1} = {2}",
                        number1, number2, number3);
   }
}
// The example displays the following output:
//        120.07 - 163.19 = -43.12
```

If the language you're using doesn't support custom operators, call the Subtract method instead.

The equivalent method for this operator is Decimal.Subtract(Decimal, Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(Decimal, Decimal)

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.UnaryNegation(Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Negates the value of the specified Decimal operand.

```
C#
public static decimal operator - (decimal d);
```

Parameters

d Decimal

The value to negate.

Returns

Decimal

The result of `d` multiplied by negative one (-1).

Implements

UnaryNegation(TSelf)

Remarks

The UnaryPlus method defines the operation of the unary negation operator (or the additive inverse operator) for Decimal values. The operation produces a value that results in 0 (zero) when it is added to the original number. Languages that do not support custom operators can call the Negate method instead.

The equivalent method for this operator is Decimal.Negate(Decimal)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Negate(Decimal)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.UnaryPlus(Decimal) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the value of the Decimal operand (the sign of the operand is unchanged).

C#

```
public static decimal operator + (decimal d);
```

Parameters

d Decimal

The operand to return.

Returns

Decimal

The value of the operand, `d`.

Implements

UnaryPlus(TSelf)

Remarks

The UnaryPlus method defines the operation of the unary positive operator for Decimal values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Decimal) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than Byte.MinValue or greater than Byte.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is

to call either the Decimal.ToByte method or the Convert.ToByte method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

None. This conversion is not supported.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

None. This conversion is not supported.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, unchanged.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToDouble method or the Convert.ToDouble method.

Because a Double has fewer significant digits than a Decimal, this operation may produce round-off errors.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

```
C#
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to a Int16.

Implements

ToInt16(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than Int16.MinValue or greater than Int16.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is

to call either the Decimal.ToInt16 method or the Convert.ToInt16 method.

The Decimal value is rounded toward zero to the nearest integer value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

The parameter is ignored.

Returns

Int32

The value of the current instance, converted to a Int32.

Implements

ToInt32(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than Int32.MinValue or greater than Int32.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is

to call either the Decimal.ToInt32 method or the Convert.ToInt32 method.

The Decimal value is rounded toward zero to the nearest integer value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to a Int64.

Implements

ToInt64(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than Int64.MinValue or greater than Int64.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is

to call either the Decimal.ToInt64 method or the Convert.ToInt64 method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to a SByte.

Implements

ToSByte(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than SByte.MinValue or greater than SByte.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToSByte method or the Convert.ToSByte method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToSingle method or the Convert.ToSingle method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The type to which to convert the value of this Decimal instance.

`provider` IFormatProvider

An IFormatProvider implementation that supplies culture-specific information about the format of the returned value.

Returns

Object

The value of the current instance, converted to a `type`.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

The requested type conversion is not supported.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than UInt16.MinValue or greater than UInt16.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToUInt16 method or the Convert.ToUInt16 method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than UInt32.MinValue or greater than UInt32.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToUInt32 method or the Convert.ToUInt32 method.

The Decimal value is rounded toward zero to the nearest integer value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Exceptions

OverflowException

The resulting integer value is less than UInt64.MinValue or greater than UInt64.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the Decimal instance is cast to an IConvertible interface. The recommended alternative is to call either the Decimal.ToUInt64 method or the Convert.ToUInt64 method.

The Decimal value is rounded toward zero to the nearest integer value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.IAdditiveIdentity<Decimal,Decimal>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static decimal
System.Numerics.IAdditiveIdentity<System.Decimal,System.Decimal>.AdditiveIde
ntity { get; }
```

Property Value

Decimal

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.IFloatingPoint<Decimal>.GetExponentByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<decimal>.GetExponentByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

Implements

GetExponentByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.IFloatingPoint<Decimal>.GetExponentShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current exponent.

C#

```
int IFloatingPoint<decimal>.GetExponentShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current exponent.

Implements

GetExponentShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.IFloatingPoint<Decimal>.GetSignificandBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the current significand.

C#

```
int IFloatingPoint<decimal>.GetSignificandBitLength ();
```

Returns

Int32

The length, in bits, of the current significand.

Implements

GetSignificandBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Decimal.IFloatingPoint<Decimal>.GetSignificandByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<decimal>.GetSignificandByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

Implements

GetSignificandByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.IFloatingPoint<Decimal>.TryWriteExponentBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<decimal>.TryWriteExponentBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IFloatingPoint<Decimal>.TryWriteExponentLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<decimal>.TryWriteExponentLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IFloatingPoint<Decimal>.TryWriteSignificandBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<decimal>.TryWriteSignificandBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IFloatingPoint<Decimal>.TryWriteSignificandLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<decimal>.TryWriteSignificandLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.IFloatingPoint Constants<Decimal>.E Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `e`.

```
C#

static decimal System.Numerics.IFloatingPointConstants<System.Decimal>.E {
get; }
```

Property Value

Decimal

Implements

E

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.IFloatingPoint Constants<Decimal>.Pi Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `pi`.

```
C#

static decimal System.Numerics.IFloatingPointConstants<System.Decimal>.Pi {
get; }
```

Property Value

Decimal

Implements

Pi

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.IFloatingPoint Constants<Decimal>.Tau Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `tau`.

```
C#

static decimal System.Numerics.IFloatingPointConstants<System.Decimal>.Tau {
get; }
```

Property Value

Decimal

Implements

Tau

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.IMinMaxValue<Decimal>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static decimal System.Numerics.IMinMaxValue<System.Decimal>.MaxValue { get;
}
```

Property Value

Decimal

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.IMinMaxValue<Decimal>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static decimal System.Numerics.IMinMaxValue<System.Decimal>.MinValue { get;
}
```

Property Value

Decimal

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.IMultiplicativeIdentity<Decimal,Decimal>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static decimal
System.Numerics.IMultiplicativeIdentity<System.Decimal,System.Decimal>.Multi
plicativeIdentity { get; }
```

Property Value

Decimal

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Decimal.INumber<Decimal>.MaxNumber(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static decimal INumber<decimal>.MaxNumber (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumber<Decimal>.MinNumber(Decimal, Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static decimal INumber<decimal>.MinNumber (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsComplexNumber(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<decimal>.IsComplexNumber (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsFinite(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<decimal>.IsFinite (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsImaginaryNumber(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<decimal>.IsImaginaryNumber (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsInfinity(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<decimal>.IsInfinity (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsNaN(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<decimal>.IsNaN (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsNegativeInfinity(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<decimal>.IsNegativeInfinity (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsNormal(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<decimal>.IsNormal (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsPositiveInfinity(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<decimal>.IsPositiveInfinity (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsRealNumber(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<decimal>.IsRealNumber (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsSubnormal(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

C#

```
static bool INumberBase<decimal>.IsSubnormal (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.IsZero(Decimal) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<decimal>.IsZero (decimal value);
```

Parameters

`value` Decimal

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.MaxMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static decimal INumberBase<decimal>.MaxMagnitudeNumber (decimal x, decimal y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.MinMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static decimal INumberBase<decimal>.MinMagnitudeNumber (decimal x, decimal
y);
```

Parameters

x Decimal

The value to compare with `y`.

y Decimal

The value to compare with `x`.

Returns

Decimal

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static decimal System.Numerics.INumberBase<System.Decimal>.One { get; }
```

Property Value

Decimal

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.INumberBase<Decimal>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Decimal>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<decimal>.TryConvertFromChecked<TOther> (TOther
value, out decimal result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Decimal

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<decimal>.TryConvertFromSaturating<TOther> (TOther
value, out decimal result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Decimal

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<decimal>.TryConvertFromTruncating<TOther> (TOther
value, out decimal result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Decimal

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<decimal>.TryConvertToChecked<TOther> (decimal value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Decimal

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<decimal>.TryConvertToSaturating<TOther> (decimal
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Decimal

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<decimal>.TryConvertToTruncating<TOther> (decimal
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Decimal

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.INumberBase<Decimal>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static decimal System.Numerics.INumberBase<System.Decimal>.Zero { get; }
```

Property Value

Decimal

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Decimal.ISignedNumber<Decimal>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
static decimal System.Numerics.ISignedNumber<System.Decimal>.NegativeOne {
get; }
```

Property Value

Decimal

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Decimal.IDeserializationCallback.OnDeserialization(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Runs when the deserialization of an object has been completed.

C#

```
void IDeserializationCallback.OnDeserialization (object sender);
```

Parameters

sender Object

The object that initiated the callback. The functionality for this parameter is not currently implemented.

Implements

OnDeserialization(Object)

Exceptions

SerializationException

The Decimal object contains invalid or corrupted data.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Decimal.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo with the data needed to serialize the target object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo to populate with data.

`context` StreamingContext

The destination (see StreamingContext) for this serialization.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Delegate Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a delegate, which is a data structure that refers to a static method or to a class instance and an instance method of that class.

```C#
public abstract class Delegate : ICloneable,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → Delegate

Derived Microsoft.Aspnet.Snapin.DialogProc
Microsoft.Aspnet.Snapin.PropSheetPageProc
Microsoft.Extensions.Caching.Memory.PostEvictionDelegate
Microsoft.Extensions.DependencyInjection.ObjectFactory
Microsoft.Extensions.DependencyInjection.ObjectFactory<T>
More...

Implements ICloneable , ISerializable

Remarks

The Delegate class is the base class for delegate types. However, only the system and compilers can derive explicitly from the Delegate class or from the MulticastDelegate class. It is also not permissible to derive a new type from a delegate type. The Delegate class is not considered a delegate type; it is a class used to derive delegate types.

Most languages implement a `delegate` keyword, and compilers for those languages are able to derive from the MulticastDelegate class; therefore, users should use the `delegate` keyword provided by the language.

Note

The common language runtime provides an `Invoke` method for each delegate type, with the same signature as the delegate. You do not have to call this method explicitly from C#, Visual Basic, or Visual C++, because the compilers call it automatically. The `Invoke` method is useful in **reflection** when you want to find the signature of the delegate type.

The common language runtime provides each delegate type with `BeginInvoke` and `EndInvoke` methods, to enable asynchronous invocation of the delegate. For more information about these methods, see Calling Synchronous Methods Asynchronously.

The declaration of a delegate type establishes a contract that specifies the signature of one or more methods. A delegate is an instance of a delegate type that has references to:

- An instance method of a type and a target object assignable to that type.
- An instance method of a type, with the hidden `this` parameter exposed in the formal parameter list. The delegate is said to be an open instance delegate.
- A static method.
- A static method and a target object assignable to the first parameter of the method. The delegate is said to be closed over its first argument.

For more information on delegate binding, see the CreateDelegate(Type, Object, MethodInfo, Boolean) method overload.

When a delegate represents an instance method closed over its first argument (the most common case), the delegate stores a reference to the method's entry point and a reference to an object, called the target, which is of a type assignable to the type that defined the method. When a delegate represents an open instance method, it stores a reference to the method's entry point. The delegate signature must include the hidden `this` parameter in its formal parameter list; in this case, the delegate does not have a reference to a target object, and a target object must be supplied when the delegate is invoked.

When a delegate represents a static method, the delegate stores a reference to the method's entry point. When a delegate represents a static method closed over its first argument, the delegate stores a reference to the method's entry point and a reference to a target object assignable to the type of the method's first argument. When the delegate is invoked, the first argument of the static method receives the target object. This first argument must be a reference type.

The invocation list of a delegate is an ordered set of delegates in which each element of the list invokes exactly one of the methods represented by the delegate. An invocation list can contain duplicate methods. During an invocation, methods are invoked in the order in which they appear in the invocation list. A delegate attempts to invoke every method in its invocation list; duplicates are invoked once for each time they appear in the invocation list. Delegates are immutable; once created, the invocation list of a delegate does not change.

Delegates are referred to as multicast, or combinable, because a delegate can invoke one or more methods and can be used in combining operations.

Combining operations, such as Combine and Remove, do not alter existing delegates. Instead, such an operation returns a new delegate that contains the results of the operation, an unchanged delegate, or `null`. A combining operation returns `null` when the result of the operation is a delegate that does not reference at least one method. A combining operation returns an unchanged delegate when the requested operation has no effect.

> **Note**
>
> Managed languages use the **Combine** and **Remove** methods to implement delegate operations. Examples include the `AddHandler` and `RemoveHandler` statements in Visual Basic and the `+=` and `-=` operators on delegate types in C#.

Starting with the .NET Framework 4, generic delegate types can have variant type parameters. Contravariant type parameters can be used as parameter types of the delegate, and a covariant type parameter can be used as the return type. This feature allows generic delegate types that are constructed from the same generic type definition to be assignment-compatible if their type arguments are reference types with an inheritance relationship, as explained in Covariance and Contravariance.

> **Note**
>
> Generic delegates that are assignment-compatible because of variance are not necessarily combinable. To be combinable, the types must match exactly. For example, suppose that a class named `Derived` is derived from a class named `Base`. A delegate of type `Action<Base>` (`Action(Of Base)` in Visual Basic) can be assigned to a variable of type `Action<Derived>`, but the two delegates cannot be combined because the types do not match exactly.

If an invoked method throws an exception, the method stops executing, the exception is passed back to the caller of the delegate, and remaining methods in the invocation list are not invoked. Catching the exception in the caller does not alter this behavior.

When the signature of the methods invoked by a delegate includes a return value, the delegate returns the return value of the last element in the invocation list. When the signature includes a parameter that is passed by reference, the final value of the parameter is the result of every method in the invocation list executing sequentially and updating the parameter's value.

The closest equivalent of a delegate in C is a function pointer. A delegate can represent a static method or an instance method. When the delegate represents an instance method, the delegate stores not only a reference to the method's entry point, but also a reference to the class instance. Unlike function pointers, delegates are object oriented and type safe.

For examples, see Supplemental API remarks for System.Delegate.CreateDelegate.

Constructors

Expand table

Delegate(Object, String)	Initializes a delegate that invokes the specified instance method on the specified class instance.
Delegate(Type, String)	Initializes a delegate that invokes the specified static method from the specified class.

Properties

Expand table

Method	Gets the method represented by the delegate.
Target	Gets the class instance on which the current delegate invokes the instance method.

Methods

Expand table

Clone()	Creates a shallow copy of the delegate.

Combine(Delegate, Delegate)	Concatenates the invocation lists of two delegates.
Combine(Delegate[])	Concatenates the invocation lists of an array of delegates.
CombineImpl(Delegate)	Concatenates the invocation lists of the specified multicast (combinable) delegate and the current multicast (combinable) delegate.
CreateDelegate(Type, MethodInfo)	Creates a delegate of the specified type to represent the specified method.
CreateDelegate(Type, MethodInfo, Boolean)	Creates a delegate of the specified type to represent the specified static method, with the specified behavior on failure to bind.
CreateDelegate(Type, Object, MethodInfo)	Creates a delegate of the specified type that represents the specified static or instance method, with the specified first argument.
CreateDelegate(Type, Object, MethodInfo, Boolean)	Creates a delegate of the specified type that represents the specified static or instance method, with the specified first argument and the specified behavior on failure to bind.
CreateDelegate(Type, Object, String)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance.
CreateDelegate(Type, Object, String, Boolean)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance with the specified case-sensitivity.
CreateDelegate(Type, Object, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance, with the specified case-sensitivity and the specified behavior on failure to bind.
CreateDelegate(Type, Type, String)	Creates a delegate of the specified type that represents the specified static method of the specified class.
CreateDelegate(Type, Type, String, Boolean)	Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity.
CreateDelegate(Type, Type, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity and the specified behavior on failure to bind.
DynamicInvoke(Object[])	Dynamically invokes (late-bound) the method represented by the current delegate.
DynamicInvokeImpl(Object[])	Dynamically invokes (late-bound) the method represented by

	the current delegate.
Equals(Object)	Determines whether the specified object and the current delegate are of the same type and share the same targets, methods, and invocation list.
GetHashCode()	Returns a hash code for the delegate.
GetInvocationList()	Returns the invocation list of the delegate.
GetMethodImpl()	Gets the method represented by the current delegate.
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Not supported.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Remove(Delegate, Delegate)	Removes the last occurrence of the invocation list of a delegate from the invocation list of another delegate.
RemoveAll(Delegate, Delegate)	Removes all occurrences of the invocation list of a delegate from the invocation list of another delegate.
RemoveImpl(Delegate)	Removes the invocation list of a delegate from the invocation list of another delegate.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Operators

Expand table

Equality(Delegate, Delegate)	Determines whether the specified delegates are equal.
Inequality(Delegate, Delegate)	Determines whether the specified delegates are not equal.

Extension Methods

Expand table

GetMethod	Gets an object that represents the method represented by the

Info(Delegate)	specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MulticastDelegate
- Calling Synchronous Methods Asynchronously
- Delegates (C# Programming Guide)
- Delegates (Visual Basic)
- How to: Define and Use Delegates (C++/CLI)
- Handling and Raising Events

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new delegate.

Overloads

Expand table

Delegate(Object, String)	Initializes a delegate that invokes the specified instance method on the specified class instance.
Delegate(Type, String)	Initializes a delegate that invokes the specified static method from the specified class.

Delegate(Object, String)

Initializes a delegate that invokes the specified instance method on the specified class instance.

C#

```
protected Delegate (object target, string method);
```

Parameters

target Object

The class instance on which the delegate invokes `method`.

method String

The name of the instance method that the delegate represents.

Exceptions

ArgumentNullException

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

There was an error binding to the target method.

Remarks

This constructor cannot be used in application code. To create a delegate by specifying the name of an instance method, use an overload of the CreateDelegate method that specifies a method name and a target object. For example, the CreateDelegate(Type, Object, String) method overload creates a delegate for an instance method with a specified name.

This constructor creates delegates for instance methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

See also

- Object
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Delegate(Type, String)

Initializes a delegate that invokes the specified static method from the specified class.

```
C#

protected Delegate (Type target, string method);
```

Parameters

`target` Type

The Type representing the class that defines `method`.

`method` String

The name of the static method that the delegate represents.

Exceptions

ArgumentNullException

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`target` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`target` represents an open generic type.

Remarks

This constructor cannot be used in application code. To create a delegate by specifying the name of a static method, use an overload of the CreateDelegate method that specifies a method name but does not specify a target object. For example, the CreateDelegate(Type, Type, String) method overload creates a static delegate for a method with a specified name.

This constructor creates delegates for static methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

See also

- Type
- String

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.Method Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the method represented by the delegate.

```
C#
public System.Reflection.MethodInfo Method { get; }
```

Property Value

MethodInfo

A MethodInfo describing the method represented by the delegate.

Exceptions

MemberAccessException

The caller does not have access to the method represented by the delegate (for example, if the method is private).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MethodInfo
- GetMethodImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Delegate.Target Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the class instance on which the current delegate invokes the instance method.

```
C#
public object? Target { get; }
```

Property Value

Object

The object on which the current delegate invokes the instance method, if the delegate represents an instance method; `null` if the delegate represents a static method.

Remarks

An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

If the delegate invokes one or more instance methods, this property returns the target of the last instance method in the invocation list.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Delegate.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a shallow copy of the delegate.

C#

```
public virtual object Clone ();
```

Returns

Object

A shallow copy of the delegate.

Implements

Clone()

Remarks

The clone has the same Type, target, method, and invocation list as the original delegate.

A shallow copy creates a new instance of the same type as the original object, and then copies the nonstatic fields of the original object. If the field is a value type, a bit-by-bit copy of the field is performed. If the field is a reference type, the reference is copied but the referred object is not; therefore, the reference in the original object and the reference in the clone point to the same object. In contrast, a deep copy of an object duplicates everything directly or indirectly referenced by the fields in the object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Delegate.Combine Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Concatenates the invocation lists of the specified multicast (combinable) delegates.

Overloads

Expand table

Combine(Delegate[])	Concatenates the invocation lists of an array of delegates.
Combine(Delegate, Delegate)	Concatenates the invocation lists of two delegates.

Combine(Delegate[])

Concatenates the invocation lists of an array of delegates.

C#

```
public static Delegate? Combine (params Delegate?[]? delegates);
```

Parameters

delegates Delegate[]

The array of delegates to combine.

Returns

Delegate

A new delegate with an invocation list that concatenates the invocation lists of the delegates in the `delegates` array. Returns `null` if `delegates` is `null`, if `delegates` contains zero elements, or if every entry in `delegates` is `null`.

Exceptions

ArgumentException

Not all the non-null entries in `delegates` are instances of the same delegate type.

Remarks

If the `delegates` array contains entries that are `null`, those entries are ignored.

The invocation list can contain duplicate entries; that is, entries that refer to the same method on the same object.

> **Note**
>
> Generic delegates that are assignment-compatible because of variance are not necessarily combinable. To be combinable, the types must match exactly. For example, suppose that a class named `Derived` is derived from a class named `Base`. A delegate of type `Action<Base>` (`Action(Of Base)` in Visual Basic) can be assigned to a variable of type `Action<Derived>`, as explained in **Covariance and Contravariance**, but the two delegates cannot be combined because the types do not match exactly.

Combine is useful for creating event handlers that call multiple methods each time an event occurs.

See also

- MulticastDelegate
- CombineImpl(Delegate)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Combine(Delegate, Delegate)

Concatenates the invocation lists of two delegates.

C#

```
public static Delegate? Combine (Delegate? a, Delegate? b);
```

Parameters

a Delegate

The delegate whose invocation list comes first.

b Delegate

The delegate whose invocation list comes last.

Returns

Delegate

A new delegate with an invocation list that concatenates the invocation lists of `a` and `b` in that order. Returns `a` if `b` is `null`, returns `b` if `a` is a null reference, and returns a null reference if both `a` and `b` are null references.

Exceptions

ArgumentException

Both `a` and `b` are not `null`, and `a` and `b` are not instances of the same delegate type.

Remarks

The invocation list can contain duplicate entries; that is, entries that refer to the same method on the same object.

> **Note**
>
> Generic delegates that are assignment-compatible because of variance are not necessarily combinable. To be combinable, the types must match exactly. For example, suppose that a class named `Derived` is derived from a class named `Base`. A delegate of type `Action<Base>` (`Action(Of Base)` in Visual Basic) can be assigned to a variable of type `Action<Derived>`, as explained in **Covariance and Contravariance**, but the two delegates cannot be combined because the types do not match exactly.

Combine is useful for creating event handlers that call multiple methods each time an event occurs.

See also

- MulticastDelegate
- CombineImpl(Delegate)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Delegate.CombineImpl(Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Concatenates the invocation lists of the specified multicast (combinable) delegate and the current multicast (combinable) delegate.

C#

```
protected virtual Delegate CombineImpl (Delegate? d);
```

Parameters

d Delegate

The multicast (combinable) delegate whose invocation list to append to the end of the invocation list of the current multicast (combinable) delegate.

Returns

Delegate

A new multicast (combinable) delegate with an invocation list that concatenates the invocation list of the current multicast (combinable) delegate and the invocation list of `d`, or the current multicast (combinable) delegate if `d` is `null`.

Exceptions

MulticastNotSupportedException

Always thrown.

Remarks

This method applies only if the current delegate is multicast (combinable).

The current implementation simply throws a MulticastNotSupportedException.

The invocation list can contain duplicate entries; that is, entries that refer to the same method on the same object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Combine(Delegate, Delegate)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.CreateDelegate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a delegate of the specified type.

Overloads

Expand table

CreateDelegate(Type, Type, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity and the specified behavior on failure to bind.
CreateDelegate(Type, Type, String, Boolean)	Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity.
CreateDelegate(Type, Object, String, Boolean)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance with the specified case-sensitivity.
CreateDelegate(Type, Object, MethodInfo, Boolean)	Creates a delegate of the specified type that represents the specified static or instance method, with the specified first argument and the specified behavior on failure to bind.
CreateDelegate(Type, Object, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance, with the specified case-sensitivity and the specified behavior on failure to bind.
CreateDelegate(Type, MethodInfo, Boolean)	Creates a delegate of the specified type to represent the specified static method, with the specified behavior on failure to bind.
CreateDelegate(Type, Object, String)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance.
CreateDelegate(Type, Object, MethodInfo)	Creates a delegate of the specified type that represents the specified static or instance method, with the specified first

	argument.
CreateDelegate(Type, Type, String)	Creates a delegate of the specified type that represents the specified static method of the specified class.
CreateDelegate(Type, MethodInfo)	Creates a delegate of the specified type to represent the specified method.

CreateDelegate(Type, Type, String, Boolean, Boolean)

Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity and the specified behavior on failure to bind.

```
C#

public static Delegate? CreateDelegate (Type type, Type target, string
method, bool ignoreCase, bool throwOnBindFailure);
```

Parameters

`type` Type

The Type of delegate to create.

`target` Type

The Type representing the class that implements `method`.

`method` String

The name of the static method that the delegate is to represent.

`ignoreCase` Boolean

A Boolean indicating whether to ignore the case when comparing the name of the method.

`throwOnBindFailure` Boolean

`true` to throw an exception if `method` cannot be bound; otherwise, `false`.

Returns

Delegate

A delegate of the specified type that represents the specified static method of the specified class.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`target` is not a `RuntimeType`.

-or-

`target` is an open generic type. That is, its ContainsGenericParameters property is `true`.

-or-

`method` is not a `static` method (`Shared` method in Visual Basic).

-or-

`method` cannot be bound, for example because it cannot be found, and `throwOnBindFailure` is `true`.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for static methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Type, String, Boolean)

Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity.

C#

```
public static Delegate CreateDelegate (Type type, Type target, string
method, bool ignoreCase);
```

Parameters

type Type

The Type of delegate to create.

`target` Type

The Type representing the class that implements `method`.

`method` String

The name of the static method that the delegate is to represent.

`ignoreCase` Boolean

A Boolean indicating whether to ignore the case when comparing the name of the method.

Returns

Delegate

A delegate of the specified type that represents the specified static method of the specified class.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`target` is not a `RuntimeType`.

-or-

`target` is an open generic type. That is, its ContainsGenericParameters property is `true`.

-or-

`method` is not a `static` method (`Shared` method in Visual Basic).

-or-

`method` cannot be bound, for example because it cannot be found.

MissingMethodException
The `Invoke` method of `type` is not found.

MethodAccessException
The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for static methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

This method overload is equivalent to calling the CreateDelegate(Type, Type, String, Boolean, Boolean) method overload, specifying `true` for `throwOnBindFailure`.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

CreateDelegate(Type, Object, String, Boolean)

Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance with the specified case-sensitivity.

C#

```
public static Delegate CreateDelegate (Type type, object target, string
method, bool ignoreCase);
```

Parameters

`type` Type

The Type of delegate to create.

`target` Object

The class instance on which `method` is invoked.

`method` String

The name of the instance method that the delegate is to represent.

`ignoreCase` Boolean

A Boolean indicating whether to ignore the case when comparing the name of the method.

Returns

Delegate

A delegate of the specified type that represents the specified instance method to invoke on the specified class instance.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` is not an instance method.

-or-

`method` cannot be bound, for example because it cannot be found.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for instance methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

This method overload is equivalent to calling the CreateDelegate(Type, Object, String, Boolean, Boolean) method overload, specifying `true` for `throwOnBindFailure`.

See also

- Type
- MulticastDelegate
- ReflectionPermission

- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Object, MethodInfo, Boolean)

Creates a delegate of the specified type that represents the specified static or instance method, with the specified first argument and the specified behavior on failure to bind.

C#

```
public static Delegate? CreateDelegate (Type type, object? firstArgument,
System.Reflection.MethodInfo method, bool throwOnBindFailure);
```

Parameters

`type` Type

A Type representing the type of delegate to create.

`firstArgument` Object

An Object that is the first argument of the method the delegate represents. For instance methods, it must be compatible with the instance type.

`method` MethodInfo

The MethodInfo describing the static or instance method the delegate is to represent.

`throwOnBindFailure` Boolean

`true` to throw an exception if `method` cannot be bound; otherwise, `false`.

Returns

Delegate

A delegate of the specified type that represents the specified static or instance method, or `null` if `throwOnBindFailure` is `false` and the delegate cannot be bound to `method`.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` cannot be bound, and `throwOnBindFailure` is `true`.

-or-

`method` is not a `RuntimeMethodInfo`. See Runtime Types in Reflection.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This method overload and the CreateDelegate(Type, Object, MethodInfo) method overload, which always throws on failure to bind, provide the most flexible way to create delegates. You can use them to create delegates for either static or instance methods, with or without a first argument.

> ⓘ **Note**
>
> If you do not supply a first argument, use the **CreateDelegate(Type, MethodInfo, Boolean)** method overload for better performance.

For more information and examples, see Supplemental API remarks for System.Delegate.CreateDelegate.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- MethodInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Object, String, Boolean, Boolean)

Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance, with the specified case-sensitivity and the specified behavior on failure to bind.

```
C#
```

```
public static Delegate? CreateDelegate (Type type, object target, string
method, bool ignoreCase, bool throwOnBindFailure);
```

Parameters

`type` Type

The Type of delegate to create.

`target` Object

The class instance on which `method` is invoked.

`method` String

The name of the instance method that the delegate is to represent.

`ignoreCase` Boolean

A Boolean indicating whether to ignore the case when comparing the name of the method.

`throwOnBindFailure` Boolean

`true` to throw an exception if `method` cannot be bound; otherwise, `false`.

Returns

Delegate

A delegate of the specified type that represents the specified instance method to invoke on the specified class instance.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` is not an instance method.

-or-

`method` cannot be bound, for example because it cannot be found, and `throwOnBindFailure` is `true`.

MissingMethodException
The `Invoke` method of `type` is not found.

MethodAccessException
The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for instance methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

CreateDelegate(Type, MethodInfo, Boolean)

Creates a delegate of the specified type to represent the specified static method, with the specified behavior on failure to bind.

```
C#

public static Delegate? CreateDelegate (Type type,
System.Reflection.MethodInfo method, bool throwOnBindFailure);
```

Parameters

`type` Type

The Type of delegate to create.

`method` MethodInfo

The MethodInfo describing the static or instance method the delegate is to represent.

`throwOnBindFailure` Boolean

`true` to throw an exception if `method` cannot be bound; otherwise, `false`.

Returns

Delegate

A delegate of the specified type to represent the specified static method.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` cannot be bound, and `throwOnBindFailure` is `true`.

-or-

`method` is not a `RuntimeMethodInfo`. See Runtime Types in Reflection.

MissingMethodException
The `Invoke` method of `type` is not found.

MethodAccessException
The caller does not have the permissions necessary to access `method`.

Remarks

This method overload can create open static method delegates and open instance method delegates - that is, delegates that expose the hidden first argument of instance methods. For a detailed explanation, see the more general CreateDelegate(Type, Object, MethodInfo, Boolean) method overload, which allows you to create all combinations of open or closed delegates for instance or static methods.

> **Note**
>
> This method overload should be used when the delegate is not closed over its first argument, because it is somewhat faster in that case.

For examples, see Supplemental API remarks for System.Delegate.CreateDelegate.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- MethodInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Object, String)

Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance.

C#

```
public static Delegate CreateDelegate (Type type, object target, string
method);
```

Parameters

`type` Type

The Type of delegate to create.

`target` Object

The class instance on which `method` is invoked.

`method` String

The name of the instance method that the delegate is to represent.

Returns

Delegate

A delegate of the specified type that represents the specified instance method to invoke on the specified class instance.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` is not an instance method.

-or-

`method` cannot be bound, for example because it cannot be found.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for instance methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

This method overload is equivalent to calling the CreateDelegate(Type, Object, String, Boolean, Boolean) method overload, specifying `false` for `ignoreCase` and `true` for `throwOnBindFailure`.

See also

- Type

- MulticastDelegate
- ReflectionPermission
- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Object, MethodInfo)

Creates a delegate of the specified type that represents the specified static or instance method, with the specified first argument.

C#

```
public static Delegate CreateDelegate (Type type, object? firstArgument,
System.Reflection.MethodInfo method);
```

Parameters

`type` Type

The Type of delegate to create.

`firstArgument` Object

The object to which the delegate is bound, or `null` to treat `method` as `static` (`Shared` in Visual Basic).

`method` MethodInfo

The MethodInfo describing the static or instance method the delegate is to represent.

Returns

Delegate

A delegate of the specified type that represents the specified static or instance method.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` cannot be bound.

-or-

`method` is not a `RuntimeMethodInfo`. See Runtime Types in Reflection.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

Calling this method overload is equivalent to calling the CreateDelegate(Type, Object, MethodInfo, Boolean) method overload and specifying `true` for `throwOnBindFailure`. These two overloads provide the most flexible way to create delegates. You can use them to create delegates for either static or instance methods, and optionally to specify the first argument.

> ⓘ **Note**
>
> If you don't supply a first argument, use the **CreateDelegate(Type, MethodInfo)** method overload for better performance.

For more information about this API, see Supplemental API remarks for System.Delegate.CreateDelegate.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- MethodInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, Type, String)

Creates a delegate of the specified type that represents the specified static method of the specified class.

C#

```
public static Delegate CreateDelegate (Type type, Type target, string
method);
```

Parameters

type Type

The Type of delegate to create.

`target` Type

The Type representing the class that implements `method`.

`method` String

The name of the static method that the delegate is to represent.

Returns

Delegate

A delegate of the specified type that represents the specified static method of the specified class.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`target` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`target` is not a `RuntimeType`.

-or-

`target` is an open generic type. That is, its ContainsGenericParameters property is `true`.

-or-

`method` is not a `static` method (`Shared` method in Visual Basic).

-or-

`method` cannot be bound, for example because it cannot be found, and `throwOnBindFailure` is `true`.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This method creates delegates for static methods only. An instance method is a method that is associated with an instance of a class; a static method is a method that is associated with the class itself.

This method overload is equivalent to calling the CreateDelegate(Type, Type, String, Boolean, Boolean) method overload, specifying `false` for `ignoreCase` and `true` for `throwOnBindFailure`.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateDelegate(Type, MethodInfo)

Creates a delegate of the specified type to represent the specified method.

```
C#

public static Delegate CreateDelegate (Type type,
System.Reflection.MethodInfo method);
```

Parameters

`type` Type

The Type of delegate to create.

`method` MethodInfo

The MethodInfo describing the static or instance method the delegate is to represent.

Returns

Delegate

A delegate of the specified type to represent the specified method.

Exceptions

ArgumentNullException

`type` is `null`.

-or-

`method` is `null`.

ArgumentException

`type` does not inherit MulticastDelegate.

-or-

`type` is not a `RuntimeType`. See Runtime Types in Reflection.

-or-

`method` cannot be bound.

-or-

`method` is not a `RuntimeMethodInfo`. See Runtime Types in Reflection.

MissingMethodException

The `Invoke` method of `type` is not found.

MethodAccessException

The caller does not have the permissions necessary to access `method`.

Remarks

This overload can create delegates for static methods and open instance method delegates; that is, delegates that explicitly supply the hidden first argument of instance methods. For a detailed explanation, see the more general CreateDelegate(Type, Object, MethodInfo) method overload, which allows you to create all combinations of open or closed delegates for instance or static methods, and optionally to specify a first argument.

> **Note**
>
> This method overload should be used when the delegate is not closed over its first argument, because it is somewhat faster in that case.

For more information about this API, see Supplemental API remarks for System.Delegate.CreateDelegate.

See also

- Type
- MulticastDelegate
- ReflectionPermission
- MethodInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.DynamicInvoke(Object[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Dynamically invokes (late-bound) the method represented by the current delegate.

C#

```
public object? DynamicInvoke (params object?[]? args);
```

Parameters

args Object[]

An array of objects that are the arguments to pass to the method represented by the current delegate.

-or-

`null`, if the method represented by the current delegate does not require arguments.

Returns

Object

The object returned by the method represented by the delegate.

Exceptions

TargetParameterCountException

The `args` array does not have the correct number of arguments.

ArgumentException

The elements of the `args` array do not match the signature of the delegate.

TargetInvocationException

The method represented by the delegate is an instance method and the target object is `null`.

-or-

One of the encapsulated methods throws an exception.

MemberAccessException
The caller does not have access to the method represented by the delegate (for example, if the method is private).

-or-

The number, order, or type of parameters listed in `args` is invalid.

Remarks

This method calls the DynamicInvokeImpl method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- DynamicInvokeImpl(Object[])
- Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Delegate.DynamicInvokeImpl(Object[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Dynamically invokes (late-bound) the method represented by the current delegate.

C#

```
protected virtual object? DynamicInvokeImpl (object?[]? args);
```

Parameters

args Object[]

An array of objects that are the arguments to pass to the method represented by the current delegate.

-or-

`null`, if the method represented by the current delegate does not require arguments.

Returns

Object

The object returned by the method represented by the delegate.

Exceptions

TargetParameterCountException

The `args` array does not have the correct number of arguments.

ArgumentException

The elements of the `args` array do not match the signature of the delegate.

TargetInvocationException

The method represented by the delegate is an instance method and the target object is `null`.

-or-

One of the encapsulated methods throws an exception.

MemberAccessException
The caller does not have access to the method represented by the delegate (for example, if the method is private).

-or-

The number, order, or type of parameters listed in `args` is invalid.

Remarks

This method implements the DynamicInvoke method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- DynamicInvoke(Object[])
- Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Delegate.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified object and the current delegate are of the same type and share the same targets, methods, and invocation list.

```C#
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with the current delegate.

Returns

Boolean

`true` if `obj` and the current delegate have the same targets, methods, and invocation list; otherwise, `false`.

Exceptions

MemberAccessException

The caller does not have access to the method represented by the delegate (for example, if the method is private).

Remarks

If the two delegates are not of the same type, they are not considered equal.

The methods and targets are compared for equality as follows:

- If the two methods being compared are both static and are the same method on the same class, the methods are considered equal and the targets are also considered equal.
- If the two methods being compared are instance methods and are the same method on the same object, the methods are considered equal and the targets are also considered equal.
- Otherwise, the methods are not considered to be equal and the targets are also not considered to be equal.

Two invocation lists are considered identical only if they have the same order and the corresponding elements from the two lists represent the same method and target.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a hash code for the delegate.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A hash code for the delegate.

Remarks

The return value of this method must not be persisted for two reasons. First, the hash function of a class might be altered to generate a better distribution, rendering any values from the old hash function useless. Second, the default implementation of this class does not guarantee that the same value will be returned by different instances.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetHashCode()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.GetInvocationList Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the invocation list of the delegate.

C#

```
public virtual Delegate[] GetInvocationList ();
```

Returns

Delegate[]

An array of delegates representing the invocation list of the current delegate.

Examples

The following example assigns three methods to a delegate. It then calls the GetInvocationList method to get a total count of the methods assigned to the delegate, to execute the delegates in reverse order, and to execute the methods whose name do not include the substring "File".

C#

```
using System;
using System.IO;
using System.Reflection;
using System.Windows.Forms;

public class Example
{
   public static void Main()
   {
      Action<String> outputMessage = null;
      outputMessage += Console.WriteLine;
      outputMessage += OutputToFile;
      outputMessage += ShowMessageBox;

//       Dim output1 As Action(Of String) = AddressOf Console.WriteLine
```

```
//       Dim output2 As Action(Of String) = AddressOf OutputToFile
//       Dim output3 As Action(Of String) = AddressOf MessageBox.Show
//
//       outputMessage = [Delegate].Combine( { output1, output2, output3 } )
      Console.WriteLine("Invocation list has {0} methods.",
                        outputMessage.GetInvocationList().Length);

      // Invoke delegates normally.
      outputMessage("Hello there!");
      Console.WriteLine("Press <Enter> to continue...");
      Console.ReadLine();

      // Invoke each delegate in the invocation list in reverse order.
      for (int ctr = outputMessage.GetInvocationList().Length - 1; ctr >= 0;
ctr--) {
         var outputMsg = outputMessage.GetInvocationList()[ctr];
         outputMsg.DynamicInvoke("Greetings and salutations!");
      }
      Console.WriteLine("Press <Enter> to continue...");
      Console.ReadLine();

      // Invoke each delegate that doesn't write to a file.
      for (int ctr = 0; ctr < outputMessage.GetInvocationList().Length;
ctr++) {
         var outputMsg = outputMessage.GetInvocationList()[ctr];
         if (! outputMsg.GetMethodInfo().Name.Contains("File"))
            outputMsg.DynamicInvoke( new String[] { "Hi!" } );
      }
   }

   private static void OutputToFile(String s)
   {
      var sw = new StreamWriter(@".\output.txt");
      sw.WriteLine(s);
      sw.Close();
   }

   private static void ShowMessageBox(String s)
   {
      MessageBox.Show(s);
   }
}
```

Remarks

Each delegate in the array represents exactly one method.

The order of the delegates in the array is the same order in which the current delegate invokes the methods that those delegates represent.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.GetMethodImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the method represented by the current delegate.

C#

```
protected virtual System.Reflection.MethodInfo GetMethodImpl ();
```

Returns

MethodInfo

A MethodInfo describing the method represented by the current delegate.

Exceptions

MemberAccessException

The caller does not have access to the method represented by the delegate (for example, if the method is private).

Remarks

This method implements the Method property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MethodInfo
- Method

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Not supported.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public virtual void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

Not supported.

`context` StreamingContext

Not supported.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

This method is not supported.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ISerializable
- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.Remove(Delegate, Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes the last occurrence of the invocation list of a delegate from the invocation list of another delegate.

```
C#

public static Delegate? Remove (Delegate? source, Delegate? value);
```

Parameters

`source` Delegate

The delegate from which to remove the invocation list of `value`.

`value` Delegate

The delegate that supplies the invocation list to remove from the invocation list of `source`.

Returns

Delegate

A new delegate with an invocation list formed by taking the invocation list of `source` and removing the last occurrence of the invocation list of `value`, if the invocation list of `value` is found within the invocation list of `source`. Returns `source` if `value` is `null` or if the invocation list of `value` is not found within the invocation list of `source`. Returns a null reference if the invocation list of `value` is equal to the invocation list of `source` or if `source` is a null reference.

Exceptions

ArgumentException

The delegate types do not match.

MemberAccessException
The caller does not have access to the method represented by the delegate (for example, if the method is private).

Remarks

If the invocation list of `value` matches a contiguous set of elements in the invocation list of `source`, then the invocation list of `value` is said to occur within the invocation list of `source`. If the invocation list of `value` occurs more than once in the invocation list of `source`, the last occurrence is removed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- RemoveAll(Delegate, Delegate)
- RemoveImpl(Delegate)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Delegate.RemoveAll(Delegate, Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes all occurrences of the invocation list of a delegate from the invocation list of another delegate.

C#

```
public static Delegate? RemoveAll (Delegate? source, Delegate? value);
```

Parameters

`source` Delegate

The delegate from which to remove the invocation list of `value`.

`value` Delegate

The delegate that supplies the invocation list to remove from the invocation list of `source`.

Returns

Delegate

A new delegate with an invocation list formed by taking the invocation list of `source` and removing all occurrences of the invocation list of `value`, if the invocation list of `value` is found within the invocation list of `source`. Returns `source` if `value` is `null` or if the invocation list of `value` is not found within the invocation list of `source`. Returns a null reference if the invocation list of `value` is equal to the invocation list of `source`, if `source` contains only a series of invocation lists that are equal to the invocation list of `value`, or if `source` is a null reference.

Exceptions

ArgumentException

The delegate types do not match.

MemberAccessException

The caller does not have access to the method represented by the delegate (for example, if the method is private).

Remarks

If the invocation list of `value` matches a contiguous set of elements in the invocation list of `source`, then the invocation list of `value` is said to occur within the invocation list of `source`. If the invocation list of `value` occurs more than once in the invocation list of `source`, all occurrences are removed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Remove(Delegate, Delegate)
- RemoveImpl(Delegate)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Delegate.RemoveImpl(Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes the invocation list of a delegate from the invocation list of another delegate.

```
C#
protected virtual Delegate? RemoveImpl (Delegate d);
```

Parameters

d Delegate

The delegate that supplies the invocation list to remove from the invocation list of the current delegate.

Returns

Delegate

A new delegate with an invocation list formed by taking the invocation list of the current delegate and removing the invocation list of `value`, if the invocation list of `value` is found within the current delegate's invocation list. Returns the current delegate if `value` is `null` or if the invocation list of `value` is not found within the current delegate's invocation list. Returns `null` if the invocation list of `value` is equal to the current delegate's invocation list.

Exceptions

MemberAccessException

The caller does not have access to the method represented by the delegate (for example, if the method is private).

Remarks

If the invocation list of `value` matches a contiguous set of elements in the current delegate's invocation list, then the invocation list of `value` is said to occur within the current delegate's invocation list. If the invocation list of `value` occurs more than once in the current delegate's invocation list, the last occurrence is removed.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Remove(Delegate, Delegate)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.Equality(Delegate, Delegate) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified delegates are equal.

C#

```
public static bool operator == (Delegate? d1, Delegate? d2);
```

Parameters

d1 Delegate

The first delegate to compare.

d2 Delegate

The second delegate to compare.

Returns

Boolean

`true` if `d1` is equal to `d2`; otherwise, `false`.

Remarks

Two delegates of the same type with the same targets, methods, and invocation lists are considered equal.

If the two delegates are not of the same type, they are not considered equal.

The methods and targets are compared for equality as follows:

- If the two methods being compared are both static and are the same method on the same class, the methods are considered equal and the targets are also

considered equal.

- If the two methods being compared are instance methods and are the same method on the same object, the methods are considered equal and the targets are also considered equal.
- Otherwise, the methods are not considered to be equal and the targets are also not considered to be equal.

Two invocation lists are considered identical if they have the same order and the corresponding elements from the two lists represent the same method and target.

The equivalent method for this operator is Delegate.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Delegate.Inequality(Delegate, Delegate) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified delegates are not equal.

```
C#
public static bool operator != (Delegate? d1, Delegate? d2);
```

Parameters

d1 Delegate

The first delegate to compare.

d2 Delegate

The second delegate to compare.

Returns

Boolean

`true` if `d1` is not equal to `d2`; otherwise, `false`.

Remarks

Two delegates are considered not equal if they are of different types, or have different methods, different targets, or different invocation lists.

If the two delegates are not of the same type, they are not considered equal.

The methods and targets are compared for equality as follows:

- If the two methods being compared are both static and are the same method on the same class, the methods are considered equal and the targets are also

considered equal.

- If the two methods being compared are instance methods and are the same method on the same object, the methods are considered equal and the targets are also considered equal.
- Otherwise, the methods are not considered to be equal and the targets are also not considered to be equal.

Two invocation lists are not equal if they have different sizes, if they are ordered differently, or if at least one element from one list represents a method or target that is different from that represented by its corresponding element in the other list.

The equivalent method for this operator is Delegate.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DivideByZeroException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to divide an integral or Decimal value by zero.

C#

```
public class DivideByZeroException : ArithmeticException
```

Inheritance Object → Exception → SystemException → ArithmeticException → DivideByZeroException

Examples

The following example handles a DivideByZeroException exception in integer division.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int number1 = 3000;
      int number2 = 0;
      try {
         Console.WriteLine(number1 / number2);
      }
      catch (DivideByZeroException) {
         Console.WriteLine("Division of {0} by zero.", number1);
      }
   }
}
// The example displays the following output:
//        Division of 3000 by zero.
```

Remarks

Trying to divide an integer or Decimal number by zero throws a DivideByZeroException exception. To prevent the exception, ensure that the denominator in a division operation with integer or Decimal values is non-zero.

Dividing a floating-point value by zero doesn't throw an exception; it results in positive infinity, negative infinity, or not a number (NaN), according to the rules of IEEE 754 arithmetic. Because the following example uses floating-point division rather than integer division, the operation does not throw a DivideByZeroException exception.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int number1 = 3000;
      int number2 = 0;

      Console.WriteLine((double)number1 / number2);
   }
}
// The example displays the following output:
//        Infinity
```

For more information, see Single and Double.

The following Microsoft intermediate language (MSIL) instructions throw DivideByZeroException:

- div
- div.un
- rem
- rem.un

DivideByZeroException uses the HRESULT COR_E_DIVIDEBYZERO, which has the value 0x80020012.

For a list of initial property values for an instance of DivideByZeroException, see the DivideByZeroException constructors.

Constructors

Expand table

DivideByZeroException()	Initializes a new instance of the DivideByZeroException class.
DivideByZero Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DivideByZeroException class with serialized data.
DivideByZeroException(String)	Initializes a new instance of the DivideByZeroException class with a specified error message.
DivideByZeroException(String, Exception)	Initializes a new instance of the DivideByZeroException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DivideByZeroException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the DivideByZeroException class.

Overloads

Expand table

DivideByZeroException()	Initializes a new instance of the DivideByZeroException class.
DivideByZeroException(String)	Initializes a new instance of the DivideByZeroException class with a specified error message.
DivideByZeroException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DivideByZeroException class with serialized data.
DivideByZeroException(String, Exception)	Initializes a new instance of the DivideByZeroException class with a specified error message and a reference to the inner exception that is the cause of this exception.

DivideByZeroException()

Initializes a new instance of the DivideByZeroException class.

C#

```
public DivideByZeroException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to divide by zero." This message takes into account the current system culture.

The following table shows the initial property values for an instance of DivideByZeroException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DivideByZeroException(String)

Initializes a new instance of the DivideByZeroException class with a specified error message.

C#

```
public DivideByZeroException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of DivideByZeroException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DivideByZeroException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the DivideByZeroException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected DivideByZeroException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DivideByZeroException(String, Exception)

Initializes a new instance of the DivideByZeroException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public DivideByZeroException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of DivideByZeroException.

Expand table

Property Type	Condition
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DllNotFoundException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.InteropServices.dll

The exception that is thrown when a DLL specified in a DLL import cannot be found.

C#

```
public class DllNotFoundException : TypeLoadException
```

Inheritance Object → Exception → SystemException → TypeLoadException → DllNotFoundException

Remarks

DllNotFoundException uses the HRESULT COR_E_DLLNOTFOUND, which has the value 0x80131524.

For a list of initial property values for an instance of DllNotFoundException, see the DllNotFoundException constructors.

The DllNotFoundException is thrown when a dynamic link library defined by the System.Runtime.InteropServices.DllImportAttribute attribute or a comparable language construct, such as the Visual Basic `Declare` statement, cannot be found. It is not thrown when the common language runtime is unable to load a .NET Framework assembly.

Note that the DllImportAttribute can specify the path to the directory in which the DLL resides. If a path is not specified, Windows uses the search order described in Dynamic-Link Library Search Order.

Constructors

Expand table

DllNotFoundException()	Initializes a new instance of the DllNotFoundException class

	with default properties.
DllNotFoundException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DllNotFoundException class with serialized data.
DllNotFoundException(String)	Initializes a new instance of the DllNotFoundException class with a specified error message.
DllNotFoundException(String, Exception)	Initializes a new instance of the DllNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message for this exception. (Inherited from TypeLoadException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)
TypeName	Gets the fully qualified name of the type that causes the exception. (Inherited from TypeLoadException)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, method name, resource ID, and additional exception information. (Inherited from TypeLoadException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DllNotFoundException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.InteropServices.dll

Initializes a new instance of the DllNotFoundException class.

Overloads

Expand table

DllNotFoundException()	Initializes a new instance of the DllNotFoundException class with default properties.
DllNotFoundException(String)	Initializes a new instance of the DllNotFoundException class with a specified error message.
DllNotFoundException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DllNotFoundException class with serialized data.
DllNotFoundException(String, Exception)	Initializes a new instance of the DllNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

DllNotFoundException()

Initializes a new instance of the DllNotFoundException class with default properties.

C#

```
public DllNotFoundException ();
```

Remarks

The following table shows the initial property values for an instance of DllNotFoundException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DllNotFoundException(String)

Initializes a new instance of the DllNotFoundException class with a specified error message.

C#

```
public DllNotFoundException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

The following table shows the initial property values for an instance of DllNotFoundException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

DllNotFoundException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the DllNotFoundException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
```

```
protected DllNotFoundException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DllNotFoundException(String, Exception)

Initializes a new instance of the DllNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public DllNotFoundException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of DllNotFoundException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a double-precision floating-point number.

C#

```
public readonly struct Double : IComparable<double>, IConvertible,
IEquatable<double>, IParsable<double>, ISpanParsable<double>,
IUtf8SpanParsable<double>,
System.Numerics.IAdditionOperators<double,double,double>,
System.Numerics.IAdditiveIdentity<double,double>,
System.Numerics.IBinaryFloatingPointIeee754<double>,
System.Numerics.IBinaryNumber<double>,
System.Numerics.IBitwiseOperators<double,double,double>,
System.Numerics.IComparisonOperators<double,double,bool>,
System.Numerics.IDecrementOperators<double>,
System.Numerics.IDivisionOperators<double,double,double>,
System.Numerics.IEqualityOperators<double,double,bool>,
System.Numerics.IExponentialFunctions<double>,
System.Numerics.IFloatingPoint<double>,
System.Numerics.IFloatingPointConstants<double>,
System.Numerics.IFloatingPointIeee754<double>,
System.Numerics.IHyperbolicFunctions<double>,
System.Numerics.IIncrementOperators<double>,
System.Numerics.ILogarithmicFunctions<double>,
System.Numerics.IMinMaxValue<double>,
System.Numerics.IModulusOperators<double,double,double>,
System.Numerics.IMultiplicativeIdentity<double,double>,
System.Numerics.IMultiplyOperators<double,double,double>,
System.Numerics.INumber<double>, System.Numerics.INumberBase<double>,
System.Numerics.IPowerFunctions<double>,
System.Numerics.IRootFunctions<double>,
System.Numerics.ISignedNumber<double>,
System.Numerics.ISubtractionOperators<double,double,double>,
System.Numerics.ITrigonometricFunctions<double>,
System.Numerics.IUnaryNegationOperators<double,double>,
System.Numerics.IUnaryPlusOperators<double,double>
```

Inheritance Object → ValueType → Double

Implements IComparable , IComparable<Double> , IConvertible , IEquatable<Double> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Double> , IParsable<TSelf> , ISpanParsable<Double> , ISpanParsable<TSelf> , IAdditionOperators<Double,Double,Double> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Double,Double> , IAdditiveIdentity<TSelf,TSelf> , IBinaryFloatingPointIeee754<Double> , IBinaryNumber<Double> , IBinaryNumber<TSelf> , IBitwiseOperators<Double,Double,Double> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<Double,Double,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<Double> , IDecrementOperators<TSelf> , IDivisionOperators<Double,Double,Double> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<Double,Double,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IExponentialFunctions<Double> , IExponentialFunctions<TSelf> , IFloatingPoint<Double> , IFloatingPoint<TSelf> , IFloatingPointConstants<Double> , IFloatingPointConstants<TSelf> , IFloatingPointIeee754<Double> , IFloatingPointIeee754<TSelf> , IHyperbolicFunctions<Double> , IHyperbolicFunctions<TSelf> , IIncrementOperators<Double> , IIncrementOperators<TSelf> , ILogarithmicFunctions<Double> , ILogarithmicFunctions<TSelf> , IMinMaxValue<Double> , IModulusOperators<Double,Double,Double> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<Double,Double> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<Double,Double,Double> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Double> , INumber<TSelf> , INumberBase<Double> , INumberBase<TSelf> ,

IPowerFunctions<Double> , IPowerFunctions<TSelf> ,

IRootFunctions<Double> , IRootFunctions<TSelf> ,

ISignedNumber<Double> , ISignedNumber<TSelf> ,

ISubtractionOperators<Double,Double,Double> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

ITrigonometricFunctions<Double> , ITrigonometricFunctions<TSelf> ,

IUnaryNegationOperators<Double,Double> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Double,Double> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Double> ,

IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Double.

Fields

Expand table

E	Represents the natural logarithmic base, specified by the constant, e.
Epsilon	Represents the smallest positive Double value that is greater than zero. This field is constant.
MaxValue	Represents the largest possible value of a Double. This field is constant.
MinValue	Represents the smallest possible value of a Double. This field is constant.
NaN	Represents a value that is not a number (`NaN`). This field is constant.
Negative Infinity	Represents negative infinity. This field is constant.
NegativeZero	Represents the number negative zero (-0).
Pi	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.
PositiveInfinity	Represents positive infinity. This field is constant.
Tau	Represents the number of radians in one turn, specified by the constant, τ.

Methods

Expand table

Abs(Double)	Computes the absolute of a value.
Acos(Double)	Computes the arc-cosine of a value.
Acosh(Double)	Computes the hyperbolic arc-cosine of a value.
AcosPi(Double)	Computes the arc-cosine of a value and divides the result by `pi`.
Asin(Double)	Computes the arc-sine of a value.
Asinh(Double)	Computes the hyperbolic arc-sine of a value.
AsinPi(Double)	Computes the arc-sine of a value and divides the result by `pi`.
Atan(Double)	Computes the arc-tangent of a value.
Atan2(Double, Double)	Computes the arc-tangent of the quotient of two values.
Atan2Pi(Double, Double)	Computes the arc-tangent for the quotient of two values and divides the result by `pi`.
Atanh(Double)	Computes the hyperbolic arc-tangent of a value.
AtanPi(Double)	Computes the arc-tangent of a value and divides the result by pi.
BitDecrement(Double)	Decrements a value to the smallest value that compares less than a given value.
BitIncrement(Double)	Increments a value to the smallest value that compares greater than a given value.
Cbrt(Double)	Computes the cube-root of a value.
Ceiling(Double)	Computes the ceiling of a value.
Clamp(Double, Double, Double)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Double)	Compares this instance to a specified double-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified double-precision floating-point number.

CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.
CopySign(Double, Double)	Copies the sign of a value to the sign of another value.
Cos(Double)	Computes the cosine of a value.
Cosh(Double)	Computes the hyperbolic cosine of a value.
CosPi(Double)	Computes the cosine of a value that has been multipled by `pi`.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DegreesToRadians(Double)	Converts a given value from degrees to radians.
Equals(Double)	Returns a value indicating whether this instance and a specified Double object represent the same value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Exp(Double)	Computes `E` raised to a given power.
Exp10(Double)	Computes `10` raised to a given power.
Exp10M1(Double)	Computes `10` raised to a given power and subtracts one.
Exp2(Double)	Computes `2` raised to a given power.
Exp2M1(Double)	Computes `2` raised to a given power and subtracts one.
ExpM1(Double)	Computes `E` raised to a given power and subtracts one.
Floor(Double)	Computes the floor of a value.
FusedMultiplyAdd(Double, Double, Double)	Computes the fused multiply-add of three values.
GetHashCode()	Returns the hash code for this instance.

GetTypeCode()	Returns the TypeCode for value type Double.
Hypot(Double, Double)	Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.
Ieee754Remainder(Double, Double)	Computes the remainder of two values as specified by IEEE 754.
ILogB(Double)	Computes the integer logarithm of a value.
IsEvenInteger(Double)	Determines if a value represents an even integral number.
IsFinite(Double)	Determines whether the specified value is finite (zero, subnormal, or normal).
IsInfinity(Double)	Returns a value indicating whether the specified number evaluates to negative or positive infinity.
IsInteger(Double)	Determines if a value represents an integral value.
IsNaN(Double)	Returns a value that indicates whether the specified value is not a number (NaN).
IsNegative(Double)	Determines whether the specified value is negative.
IsNegativeInfinity(Double)	Returns a value indicating whether the specified number evaluates to negative infinity.
IsNormal(Double)	Determines whether the specified value is normal.
IsOddInteger(Double)	Determines if a value represents an odd integral number.
IsPositive(Double)	Determines if a value is positive.
IsPositiveInfinity(Double)	Returns a value indicating whether the specified number evaluates to positive infinity.
IsPow2(Double)	Determines if a value is a power of two.
IsRealNumber(Double)	Determines if a value represents a real number.
IsSubnormal(Double)	Determines whether the specified value is subnormal.
Lerp(Double, Double, Double)	Performs a linear interpolation between two values based on the given weight.
Log(Double)	Computes the natural (`base-E` logarithm of a value.
Log(Double, Double)	Computes the logarithm of a value in the specified base.
Log10(Double)	Computes the base-10 logarithm of a value.
Log10P1(Double)	Computes the base-10 logarithm of a value plus one.

Log2(Double)	Computes the log2 of a value.
Log2P1(Double)	Computes the base-2 logarithm of a value plus one.
LogP1(Double)	Computes the natural (`base-E`) logarithm of a value plus one.
Max(Double, Double)	Compares two values to compute which is greater.
MaxMagnitude(Double, Double)	Compares two values to compute which is greater.
MaxMagnitudeNumber(Double, Double)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
MaxNumber(Double, Double)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
Min(Double, Double)	Compares two values to compute which is lesser.
MinMagnitude(Double, Double)	Compares two values to compute which is lesser.
MinMagnitudeNumber(Double, Double)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
MinNumber(Double, Double)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.
Parse(String)	Converts the string representation of a number to its double-precision floating-point number equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its double-precision floating-point number equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its double-precision floating-point

	number equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.
Pow(Double, Double)	Computes a value raised to a given power.
RadiansToDegrees(Double)	Converts a given value from radians to degrees.
ReciprocalEstimate(Double)	Computes an estimate of the reciprocal of a value.
ReciprocalSqrtEstimate(Double)	Computes an estimate of the reciprocal square root of a value.
RootN(Double, Int32)	Computes the n-th root of a value.
Round(Double)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Double, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Double, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Double, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
ScaleB(Double, Int32)	Computes the product of a value and its base-radix raised to the specified power.
Sign(Double)	Computes the sign of a value.
Sin(Double)	Computes the sine of a value.
SinCos(Double)	Computes the sine and cosine of a value.
SinCosPi(Double)	Computes the sine and cosine of a value.
Sinh(Double)	Computes the hyperbolic sine of a value.
SinPi(Double)	Computes the sine of a value that has been multiplied by `pi`.
Sqrt(Double)	Computes the square-root of a value.
Tan(Double)	Computes the tangent of a value.
Tanh(Double)	Computes the hyperbolic tangent of a value.
TanPi(Double)	Computes the tangent of a value that has been multipled by `pi`.

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
Truncate(Double)	Truncates a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current double instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, Double)	Tries to convert a UTF-8 character span containing the string representation of a number to its double-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Double)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Double)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Double)	Converts the span representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Double)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Double)	Converts a character span containing the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Double)	Converts the string representation of a number to its double-precision floating-point number equivalent. A

	return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Double)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Double)	Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Operators

Expand table

Equality(Double, Double)	Returns a value that indicates whether two specified Double values are equal.
GreaterThan(Double, Double)	Returns a value that indicates whether a specified Double value is greater than another specified Double value.
GreaterThan OrEqual(Double, Double)	Returns a value that indicates whether a specified Double value is greater than or equal to another specified Double value.
Inequality(Double, Double)	Returns a value that indicates whether two specified Double values are not equal.
LessThan(Double, Double)	Returns a value that indicates whether a specified Double value is less than another specified Double value.
LessThanOrEqual(Double, Double)	Returns a value that indicates whether a specified Double value is less than or equal to another specified Double value.

Explicit Interface Implementations

Expand table

IAddition Operators<Double,Double,Double>.Addition(Double, Double)	Adds two values together to compute their sum.
IAdditiveIdentity<Double,Double>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryNumber<Double>.AllBitsSet	Gets an instance of the binary type in which all bits are set.

IBitwiseOperators<Double,Double,Double>.BitwiseAnd(Double, Double)	Computes the bitwise-and of two values.
IBitwiseOperators<Double,Double,Double>.BitwiseOr(Double, Double)	Computes the bitwise-or of two values.
IBitwiseOperators<Double,Double,Double>.ExclusiveOr(Double, Double)	Computes the exclusive-or of two values.
IBitwiseOperators<Double,Double,Double>.OnesComplement(Double)	Computes the ones-complement representation of a given value.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).

IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Double>.Decrement(Double)	Decrements a value.
IDivisionOperators<Double,Double,Double>.Division(Double, Double)	Divides one value by another to compute their quotient.
IFloatingPoint<Double>.GetExponentByteCount()	Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Double>.GetExponentShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current exponent.
IFloatingPoint<Double>.GetSignificandBitLength()	Gets the length, in bits, of the current significand.
IFloatingPoint<Double>.GetSignificandByteCount()	Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Double>.TryWriteExponentBigEndian(Span<Byte>, Int32)	Tries to write the current exponent, in big-endian format, to a given span.
IFloatingPoint<Double>.TryWriteExponentLittleEndian(Span<Byte>, Int32)	Tries to write the current exponent, in little-endian format, to a given span.
IFloatingPoint<Double>.TryWriteSignificandBigEndian(Span<Byte>, Int32)	Tries to write the current significand, in big-endian format, to a given span.
IFloatingPoint<Double>.TryWriteSignificandLittleEndian(Span<Byte>, Int32)	Tries to write the current significand, in little-endian format, to a given span.
IFloatingPointConstants<Double>.E	Gets the mathematical constant `e`.
IFloatingPointConstants<Double>.Pi	Gets the mathematical constant `pi`.

IFloatingPointConstants<Double>.Tau	Gets the mathematical constant `tau`.
IFloatingPointIeee754<Double>.Epsilon	Gets the smallest value such that can be added to `0` that does not result in `0`.
IFloatingPointIeee754<Double>.NaN	Gets a value that represents `NaN`.
IFloatingPointIeee754<Double>.NegativeInfinity	Gets a value that represents negative `infinity`.
IFloatingPointIeee754<Double>.NegativeZero	Gets a value that represents negative `zero`.
IFloatingPointIeee754<Double>.PositiveInfinity	Gets a value that represents positive `infinity`.
IIncrementOperators<Double>.Increment(Double)	Increments a value.
IMinMaxValue<Double>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Double>.MinValue	Gets the minimum value of the current type.
IModulus Operators<Double,Double,Double>.Modulus(Double, Double)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Double,Double>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiply Operators<Double,Double,Double>.Multiply(Double, Double)	Multiplies two values together to compute their product.
INumberBase<Double>.IsCanonical(Double)	Determines if a value is in its canonical representation.
INumberBase<Double>.IsComplexNumber(Double)	Determines if a value represents a complex number.
INumberBase<Double>.IsImaginaryNumber(Double)	Determines if a value represents a pure imaginary number.
INumberBase<Double>.IsZero(Double)	Determines if a value is zero.
INumberBase<Double>.One	Gets the value `1` for the type.
INumberBase<Double>.Radix	Gets the radix, or base, for the type.

INumberBase<Double>.TryConvertFromChecked<TOther>(TOther, Double)	
INumberBase<Double>.TryConvertFromSaturating<TOther>(TOther, Double)	
INumberBase<Double>.TryConvertFromTruncating<TOther>(TOther, Double)	
INumberBase<Double>.TryConvertToChecked<TOther>(Double, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Double>.TryConvertToSaturating<TOther>(Double, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Double>.TryConvertToTruncating<TOther>(Double, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Double>.Zero	Gets the value `0` for the type.
ISignedNumber<Double>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<Double,Double,Double>.Subtraction(Double, Double)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Double,Double>.UnaryNegation(Double)	Computes the unary negation of a value.
IUnaryPlusOperators<Double,Double>.UnaryPlus(Double)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Decimal
- Single
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.E Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the natural logarithmic base, specified by the constant, e.

C#

```
public const double E = 2.7182818284590451;
```

Field Value

***Value** = 2.7182818284590451*

Double

Remarks

Euler's number is approximately 2.7182818284590452354.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Epsilon Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest positive Double value that is greater than zero. This field is constant.

C#

```
public const double Epsilon = 4.94065645841247E-324;
```

Field Value

Value = 4.94065645841247E-324

Double

Remarks

For more information about this API, see Supplemental API remarks for Double.Epsilon.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of a Double. This field is constant.

C#

```
public const double MaxValue = 1.7976931348623157E+308;
```

Field Value

***Value** = 1.7976931348623157E+308*

Double

Examples

The following code example illustrates the use of Double.MaxValue:

C#

```
public class Temperature {
    public static double MinValue {
        get {
            return Double.MinValue;
        }
    }

    public static double MaxValue {
        get {
            return Double.MaxValue;
        }
    }

    // The value holder
    protected double m_value;

    public double Value {
        get {
            return m_value;
        }
```

```
        set {
            m_value = value;
        }
    }

    public double Celsius {
        get {
            return (m_value-32.0)/1.8;
        }
        set {
            m_value = 1.8*value+32.0;
        }
    }
}
```

Remarks

The value of this constant is positive 1.7976931348623157E+308.

The result of an operation that exceeds Double.MaxValue is Double.PositiveInfinity. In the following example, Double.PositiveInfinity results from addition, multiplication, and exponentiation operations when the result exceeds Double.MaxValue.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double result1 = 7.997e307 + 9.985e307;
      Console.WriteLine("{0} (Positive Infinity: {1})",
                        result1, Double.IsPositiveInfinity(result1));

      double result2 = 1.5935e250 * 7.948e110;
      Console.WriteLine("{0} (Positive Infinity: {1})",
                        result2, Double.IsPositiveInfinity(result2));

      double result3 = Math.Pow(1.256e100, 1.34e20);
      Console.WriteLine("{0} (Positive Infinity: {1})",
                        result3, Double.IsPositiveInfinity(result3));
   }
}
// The example displays the following output:
//    Infinity (Positive Infinity: True)
//    Infinity (Positive Infinity: True)
//    Infinity (Positive Infinity: True)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of a Double. This field is constant.

C#

```
public const double MinValue = -1.7976931348623157E+308;
```

Field Value

Value* = *-1.7976931348623157E+308

Double

Examples

The following code example illustrates the use of Double.MinValue:

C#

```
public class Temperature {
    public static double MinValue {
        get {
            return Double.MinValue;
        }
    }

    public static double MaxValue {
        get {
            return Double.MaxValue;
        }
    }

    // The value holder
    protected double m_value;

    public double Value {
        get {
            return m_value;
        }
```

```
        set {
            m_value = value;
        }
    }

    public double Celsius {
        get {
            return (m_value-32.0)/1.8;
        }
        set {
            m_value = 1.8*value+32.0;
        }
    }
}
```

Remarks

The value of this constant is negative 1.7976931348623157E+308.

The result of an operation that is less than Double.MinValue is Double.NegativeInfinity. In the following example, Double.NegativeInfinity results from subtraction and multiplication operations when the result is less than Double.MinValue.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double result1 = -7.997e307 + -9.985e307;
      Console.WriteLine("{0} (Negative Infinity: {1})",
                        result1, Double.IsNegativeInfinity(result1));

      double result2 = -1.5935e250 * 7.948e110;
      Console.WriteLine("{0} (Negative Infinity: {1})",
                        result2, Double.IsNegativeInfinity(result2));
   }
}
// The example displays the following output:
//    -Infinity (Negative Infinity: True)
//    -Infinity (Negative Infinity: True)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.NaN Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value that is not a number (`NaN`). This field is constant.

C#

```
public const double NaN = NaN;
```

Field Value

Value* = *NaN

Double

Examples

The following example illustrates the use of NaN:

C#

```
Double zero = 0;

// This condition will return false.
if ((0 / zero) == Double.NaN)
   Console.WriteLine("0 / 0 can be tested with Double.NaN.");
else
   Console.WriteLine("0 / 0 cannot be tested with Double.NaN; use
Double.IsNan() instead.");
```

Remarks

A method or operator returns NaN when the result of an operation is undefined. For example, the result of dividing zero by zero is NaN, as the following example shows. (But note that dividing a non-zero number by zero returns either PositiveInfinity or NegativeInfinity, depending on the sign of the divisor.)

C#

```
double zero = 0.0;
Console.WriteLine("{0} / {1} = {2}", zero, zero, zero/zero);
// The example displays the following output:
//         0 / 0 = NaN
```

In addition, a method call with a NaN value or an operation on a NaN value returns NaN, as the following example shows.

C#

```
double nan1 = Double.NaN;

Console.WriteLine("{0} + {1} = {2}", 3, nan1, 3 + nan1);
Console.WriteLine("Abs({0}) = {1}", nan1, Math.Abs(nan1));
// The example displays the following output:
//       3 + NaN = NaN
//       Abs(NaN) = NaN
```

Use the IsNaN method to determine whether a value is not a number. The Equality operator considers two NaN values to be unequal to one another. In general, Double operators cannot be used to compare Double.NaN with other Double values, although comparison methods (such as Equals and CompareTo) can. The following example illustrates the difference in behavior between Double comparison operators and methods.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("NaN == NaN: {0}", Double.NaN == Double.NaN);
      Console.WriteLine("NaN != NaN: {0}", Double.NaN != Double.NaN);
      Console.WriteLine("NaN.Equals(NaN): {0}",
Double.NaN.Equals(Double.NaN));
      Console.WriteLine("! NaN.Equals(NaN): {0}", !
Double.NaN.Equals(Double.NaN));
      Console.WriteLine("IsNaN: {0}", Double.IsNaN(Double.NaN));

      Console.WriteLine("\nNaN > NaN: {0}", Double.NaN > Double.NaN);
      Console.WriteLine("NaN >= NaN: {0}", Double.NaN >= Double.NaN);
      Console.WriteLine("NaN < NaN: {0}", Double.NaN < Double.NaN);
      Console.WriteLine("NaN < 100.0: {0}", Double.NaN < 100.0);
      Console.WriteLine("NaN <= 100.0: {0}", Double.NaN <= 100.0);
      Console.WriteLine("NaN >= 100.0: {0}", Double.NaN > 100.0);
```

```
        Console.WriteLine("NaN.CompareTo(NaN): {0}",
Double.NaN.CompareTo(Double.NaN));
        Console.WriteLine("NaN.CompareTo(100.0): {0}",
Double.NaN.CompareTo(100.0));
        Console.WriteLine("(100.0).CompareTo(Double.NaN): {0}",
(100.0).CompareTo(Double.NaN));
    }
}
// The example displays the following output:
//       NaN == NaN: False
//       NaN != NaN: True
//       NaN.Equals(NaN): True
//       ! NaN.Equals(NaN): False
//       IsNaN: True
//
//       NaN > NaN: False
//       NaN >= NaN: False
//       NaN < NaN: False
//       NaN < 100.0: False
//       NaN <= 100.0: False
//       NaN >= 100.0: False
//       NaN.CompareTo(NaN): 0
//       NaN.CompareTo(100.0): -1
//       (100.0).CompareTo(Double.NaN): 1
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNaN(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.NegativeInfinity Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents negative infinity. This field is constant.

C#

```
public const double NegativeInfinity = -Infinity;
```

Field Value

Value* = *-Infinity

Double

Examples

The following code example illustrates the use of NegativeInfinity:

C#

```
// This will equal Infinity.
Console.WriteLine("10.0 minus NegativeInfinity equals {0}.", (10.0 -
Double.NegativeInfinity).ToString());
```

Remarks

The value of this constant is the result of dividing a negative number by zero.

This constant is returned when the result of an operation is less than Double.MinValue.

Use IsNegativeInfinity to determine whether a value evaluates to negative infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNegativeInfinity(Double)
- IsInfinity(Double)
- PositiveInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.NegativeZero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number negative zero (-0).

C#

```
public const double NegativeZero = 0;
```

Field Value

Value* = *0

Double

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Pi Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.

```
C#
public const double Pi = 3.1415926535897931;
```

Field Value

Value = 3.1415926535897931

Double

Remarks

Pi is approximately 3.1415926535897932385.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.PositiveInfinity Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents positive infinity. This field is constant.

```
C#

public const double PositiveInfinity = Infinity;
```

Field Value

Value* = *Infinity

Double

Examples

The following code example illustrates the use of PositiveInfinity:

```
C#

// This will equal Infinity.
Console.WriteLine("PositiveInfinity plus 10.0 equals {0}.",
(Double.PositiveInfinity + 10.0).ToString());
```

Remarks

The value of this constant is the result of dividing a positive number by zero.

This constant is returned when the result of an operation is greater than Double.MaxValue.

Use IsPositiveInfinity to determine whether a value evaluates to positive infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsPositiveInfinity(Double)
- IsInfinity(Double)
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Tau Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of radians in one turn, specified by the constant, τ.

C#

```
public const double Tau = 6.2831853071795862;
```

Field Value

Value* = *6.2831853071795862

Double

Remarks

Tau is approximately 6.2831853071795864769.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Abs(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static double Abs (double value);
```

Parameters

`value` Double

The value for which to get its absolute.

Returns

Double

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Acos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value.

C#

```
public static double Acos (double x);
```

Parameters

x Double

The value, in radians, whose arc-cosine is to be computed.

Returns

Double

The arc-cosine of `x`.

Implements

Acos(TSelf)

Remarks

This computes `arccos(x)` in the interval `[+0, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Acosh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-cosine of a value.

C#

```
public static double Acosh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic arc-cosine is to be computed.

Returns

Double

The hyperbolic arc-cosine of `x`.

Implements

Acosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.AcosPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value and divides the result by `pi`.

```
C#
public static double AcosPi (double x);
```

Parameters

x Double

The value whose arc-cosine is to be computed.

Returns

Double

The arc-cosine of `x`, divided by `pi`.

Implements

AcosPi(TSelf)

Remarks

This computes `arccos(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Asin(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value.

C#

```
public static double Asin (double x);
```

Parameters

x Double

The value, in radians, whose arc-sine is to be computed.

Returns

Double

The arc-sine of `x`.

Implements

Asin(TSelf)

Remarks

This computes `arcsin(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Asinh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-sine of a value.

C#

```
public static double Asinh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic arc-sine is to be computed.

Returns

Double

The hyperbolic arc-sine of `x`.

Implements

Asinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.AsinPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value and divides the result by `pi`.

```
C#
public static double AsinPi (double x);
```

Parameters

x Double

The value whose arc-sine is to be computed.

Returns

Double

The arc-sine of `x`, divided by `pi`.

Implements

AsinPi(TSelf)

Remarks

This computes `arcsin(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Atan(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value.

C#

```
public static double Atan (double x);
```

Parameters

x Double

The value, in radians, whose arc-tangent is to be computed.

Returns

Double

The arc-tangent of `x`.

Implements

Atan(TSelf)

Remarks

This computes `arctan(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Atan2(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of the quotient of two values.

```
C#
public static double Atan2 (double y, double x);
```

Parameters

y Double

The y-coordinate of a point.

x Double

The x-coordinate of a point.

Returns

Double

The arc-tangent of `y` divided-by `x`.

Implements

Atan2(TSelf, TSelf)

Remarks

This computes `arctan(y / x)` in the interval `[-π, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Atan2Pi(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent for the quotient of two values and divides the result by `pi`.

```
C#
public static double Atan2Pi (double y, double x);
```

Parameters

y Double

The y-coordinate of a point.

x Double

The x-coordinate of a point.

Returns

Double

The arc-tangent of `y` divided-by `x`, divided by `pi`.

Implements

Atan2Pi(TSelf, TSelf)

Remarks

This computes `arctan(y / x) / π` in the interval `[-1, +1]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Atanh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-tangent of a value.

C#

```
public static double Atanh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic arc-tangent is to be computed.

Returns

Double

The hyperbolic arc-tangent of `x`.

Implements

Atanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.AtanPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value and divides the result by pi.

```
C#
public static double AtanPi (double x);
```

Parameters

x Double

The value whose arc-tangent is to be computed.

Returns

Double

The arc-tangent of `x`, divided by `pi`.

Implements

AtanPi(TSelf)

Remarks

This computes `arctan(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.BitDecrement(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value to the smallest value that compares less than a given value.

C#

```
public static double BitDecrement (double x);
```

Parameters

x Double

The value to be bitwise decremented.

Returns

Double

The smallest value that compares less than `x`.

Implements

BitDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.BitIncrement(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value to the smallest value that compares greater than a given value.

C#

```
public static double BitIncrement (double x);
```

Parameters

x Double

The value to be bitwise incremented.

Returns

Double

The smallest value that compares greater than `x`.

Implements

BitIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Cbrt(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cube-root of a value.

```
C#
public static double Cbrt (double x);
```

Parameters

x Double

The value whose cube-root is to be computed.

Returns

Double

The cube-root of `x`.

Implements

Cbrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Ceiling(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ceiling of a value.

C#

```
public static double Ceiling (double x);
```

Parameters

x Double

The value whose ceiling is to be computed.

Returns

Double

The ceiling of `x`.

Implements

Ceiling(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Clamp(Double, Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static double Clamp (double value, double min, double max);
```

Parameters

`value` Double

The value to clamp.

`min` Double

The inclusive minimum to which `value` should clamp.

`max` Double

The inclusive maximum to which `value` should clamp.

Returns

Double

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Double object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object or Double object.

Overloads

Expand table

CompareTo(Double)	Compares this instance to a specified double-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified double-precision floating-point number.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.

CompareTo(Double)

Compares this instance to a specified double-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified double-precision floating-point number.

C#

```
public int CompareTo (double value);
```

Parameters

`value` Double

A double-precision floating-point number to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`, or this instance is not a number (NaN) and `value` is a number.
Zero	This instance is equal to `value`, or both this instance and `value` are not a number (NaN), PositiveInfinity, or NegativeInfinity.
Greater than zero	This instance is greater than `value`, or this instance is a number and `value` is not a number (NaN).

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";
```

```
    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
```

```
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
```

```
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

For more information about this API, see Double.CompareTo.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Value	Description
A negative integer	This instance is less than `value`, or this instance is not a number (NaN) and `value` is a number.
Zero	This instance is equal to `value`, or this instance and `value` are both `Double.NaN`, PositiveInfinity, or NegativeInfinity
A positive integer	This instance is greater than `value`, OR this instance is a number and `value` is not a number (NaN), OR `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Double.

Examples

The following code example illustrates the use of `CompareTo` in the context of `Double`.

C#

```
obj1 = (Double)450;

if (a.CompareTo(obj1) < 0)
   Console.WriteLine("{0} is less than {1}.", a.ToString(),
obj1.ToString());

if (a.CompareTo(obj1) > 0)
   Console.WriteLine("{0} is greater than {1}.", a.ToString(),
obj1.ToString());

if (a.CompareTo(obj1) == 0)
   Console.WriteLine("{0} equals {1}.", a.ToString(), obj1.ToString());
```

Remarks

For more information about this API, see Double.CompareTo.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CopySign(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
public static double CopySign (double value, double sign);
```

Parameters

`value` Double

The value whose magnitude is used in the result.

`sign` Double

The value whose sign is used in the result.

Returns

Double

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Cos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value.

```
C#
public static double Cos (double x);
```

Parameters

x Double

The value, in radians, whose cosine is to be computed.

Returns

Double

The cosine of `x`.

Implements

Cos(TSelf)

Remarks

This computes `cos(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Cosh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic cosine of a value.

C#

```
public static double Cosh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic cosine is to be computed.

Returns

Double

The hyperbolic cosine of `x`.

Implements

Cosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CosPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value that has been multipled by `pi`.

```
C#
public static double CosPi (double x);
```

Parameters

x Double

The value, in half-revolutions, whose cosine is to be computed.

Returns

Double

The cosine of `x` multiplied-by `pi`.

Implements

CosPi(TSelf)

Remarks

This computes `cos(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static double CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Double

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static double CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Double

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

```
C#

public static double CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Double

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.DegreesToRadians(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from degrees to radians.

C#

```
public static double DegreesToRadians (double degrees);
```

Parameters

`degrees` Double

The value to convert to radians.

Returns

Double

The value of `degrees` converted to radians.

Implements

DegreesToRadians(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two instances of Double represent the same value.

Overloads

Expand table

Equals(Double)	Returns a value indicating whether this instance and a specified Double object represent the same value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(Double)

Returns a value indicating whether this instance and a specified Double object represent the same value.

```
C#

public bool Equals (double obj);
```

Parameters

obj Double

A Double object to compare to this instance.

Returns

Boolean

`true` if `obj` is equal to this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

For more information about this API, see Supplemental API remarks for Double.Equals.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Double is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Double) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Double value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive numeric types except for Decimal and in C#, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Double) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static double value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,16}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,18}", value.Equals(int1));
      TestObjectForEquality(int1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
```

```
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,16}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        float sng1 = 112;
        Console.WriteLine("value = sng1: {0,19}", value.Equals(sng1));
        TestObjectForEquality(sng1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (Double) = 112 (Byte): False
//
//       value = short1:             True
//       112 (Double) = 112 (Int16): False
//
//       value = int1:               True
//       112 (Double) = 112 (Int32): False
//
//       value = long1:              True
//       112 (Double) = 112 (Int64): False
//
//       value = sbyte1:             True
//       112 (Double) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (Double) = 112 (UInt16): False
//
```

```
//       value = uint1:                True
//       112 (Double) = 112 (UInt32): False
//
//       value = ulong1:               True
//       112 (Double) = 112 (UInt64): False
//
//       value = dec1:                  False
//       112 (Double) = 112 (Decimal): False
//
//       value = sng1:                 True
//       112 (Double) = 112 (Single): False
```

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of Double and equals the value of this instance; otherwise, `false`.

Remarks

The Equals method should be used with caution, because two apparently equivalent values can be unequal due to the differing precision of the two values. The following example reports that the Double value .3333 and the Double returned by dividing 1 by 3 are unequal.

```C#
// Initialize two doubles with apparently identical values
double double1 = .33333;
object double2 = (double) 1/3;
// Compare them for equality
Console.WriteLine(double1.Equals(double2));    // displays false
```

For alternatives to calling the Equals method, see the documentation for the Equals(Double) overload.

> **Note**
>
> Because **Epsilon** defines the minimum expression of a positive value whose range is near zero, the margin of difference between two similar values must be greater than **Epsilon**. Typically, it is many times greater than **Epsilon**.

The precision of floating-point numbers beyond the documented precision is specific to the implementation and version of the .NET Framework. Consequently, a comparison of two particular numbers might change between versions of the .NET Framework because the precision of the numbers' internal representation might change.

If two Double.NaN values are tested for equality by calling the Equals method, the method returns `true`. However, if two NaN values are tested for equality by using the equality operator, the operator returns `false`. When you want to determine

whether the value of a Double is not a number (NaN), an alternative is to call the IsNaN method.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Double is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Double) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Double value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive numeric types except for Decimal and in C#, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Double) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static double value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,16}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,18}", value.Equals(int1));
      TestObjectForEquality(int1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
      TestObjectForEquality(long1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,16}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
```

```
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        float sng1 = 112;
        Console.WriteLine("value = sng1: {0,19}", value.Equals(sng1));
        TestObjectForEquality(sng1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (Double) = 112 (Byte): False
//
//       value = short1:             True
//       112 (Double) = 112 (Int16): False
//
//       value = int1:               True
//       112 (Double) = 112 (Int32): False
//
//       value = long1:              True
//       112 (Double) = 112 (Int64): False
//
//       value = sbyte1:             True
//       112 (Double) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (Double) = 112 (UInt16): False
//
//       value = uint1:               True
//       112 (Double) = 112 (UInt32): False
//
//       value = ulong1:              True
//       112 (Double) = 112 (UInt64): False
//
//       value = dec1:                 False
```

```
//       112 (Double) = 112 (Decimal): False
//
//       value = sng1:                 True
//       112 (Double) = 112 (Single): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Exp(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power.

```
C#
public static double Exp (double x);
```

Parameters

x Double

The power to which `E` is raised.

Returns

Double

`E` raised to the power of `x`.

Implements

Exp(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Exp10(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power.

```
C#
public static double Exp10 (double x);
```

Parameters

x Double

The power to which `10` is raised.

Returns

Double

10^x

Implements

Exp10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Exp10M1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power and subtracts one.

```
C#
public static double Exp10M1 (double x);
```

Parameters

x Double

The power to which `10` is raised.

Returns

Double

`10ˣ - 1`

Implements

Exp10M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Exp2(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power.

```
C#
public static double Exp2 (double x);
```

Parameters

x Double

The power to which `2` is raised.

Returns

Double

2^x

Implements

Exp2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Exp2M1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power and subtracts one.

```
C#
public static double Exp2M1 (double x);
```

Parameters

x Double

The power to which `2` is raised.

Returns

Double

`2ˣ - 1`

Implements

Exp2M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ExpM1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power and subtracts one.

C#

```
public static double ExpM1 (double x);
```

Parameters

x Double

The power to which `E` is raised.

Returns

Double

$E^x - 1$

Implements

ExpM1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Floor(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the floor of a value.

C#

```
public static double Floor (double x);
```

Parameters

x Double

The value whose floor is to be computed.

Returns

Double

The floor of `x`.

Implements

Floor(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.FusedMultiplyAdd(Double, Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the fused multiply-add of three values.

```
C#

public static double FusedMultiplyAdd (double left, double right, double
addend);
```

Parameters

`left` Double

The value which `right` multiplies.

`right` Double

The value which multiplies `left`.

`addend` Double

The value that is added to the product of `left` and `right`.

Returns

Double

The result of `left` times `right` plus `addend` computed as one ternary operation.

Implements

FusedMultiplyAdd(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Double.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Double.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Hypot(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.

```
C#
public static double Hypot (double x, double y);
```

Parameters

x Double

The value to square and add to `y`.

y Double

The value to square and add to `x`.

Returns

Double

The square root of `x`-squared plus `y`-squared.

Implements

Hypot(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Ieee754Remainder(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the remainder of two values as specified by IEEE 754.

```
C#
public static double Ieee754Remainder (double left, double right);
```

Parameters

`left` Double

The value which `right` divides.

`right` Double

The value which divides `left`.

Returns

Double

The remainder of `left` divided-by `right` as specified by IEEE 754.

Implements

Ieee754Remainder(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ILogB(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the integer logarithm of a value.

C#

```
public static int ILogB (double x);
```

Parameters

x Double

The value whose integer logarithm is to be computed.

Returns

Int32

The integer logarithm of `x`.

Implements

ILogB(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsEvenInteger(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (double value);
```

Parameters

value Double

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsFinite(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is finite (zero, subnormal, or normal).

C#

```
public static bool IsFinite (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if the value is finite (zero, subnormal or normal); `false` otherwise.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsInfinity(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to negative or positive infinity.

```C#
public static bool IsInfinity (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if `d` evaluates to PositiveInfinity or NegativeInfinity; otherwise, `false`.

Implements

IsInfinity(TSelf)

Examples

The following code example illustrates the use of IsInfinity:

```C#
// This will return "true".
Console.WriteLine("IsInfinity(3.0 / 0) == {0}.", Double.IsInfinity(3.0 / 0)
? "true" : "false");
```

Remarks

Floating-point operations return PositiveInfinity or NegativeInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsPositiveInfinity(Double)
- IsNegativeInfinity(Double)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IsInteger(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral value.

```
C#
public static bool IsInteger (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` and `3.0` will return `true` while `2.2` and `3.3` will return `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsNaN(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the specified value is not a number (NaN).

C#

```
public static bool IsNaN (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if `d` evaluates to NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Examples

The following code example illustrates the use of IsNaN:

C#

```
// This will return true.
if (Double.IsNaN(0 / zero))
   Console.WriteLine("Double.IsNan() can determine whether a value is not-a-
number.");
```

Remarks

Floating-point operations return NaN to signal that result of the operation is undefined. For example, dividing 0.0 by 0.0 results in NaN.

> **Note**
>
> IsNaN returns `false` if a Double value is either PositiveInfinity or NegativeInfinity. To test for these values, use the IsInfinity, IsPositiveInfinity, and IsNegativeInfinity methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- NaN

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IsNegative(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is negative.

C#

```
public static bool IsNegative (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if the value is negative; `false` otherwise.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsNegativeInfinity(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to negative infinity.

C#

```
public static bool IsNegativeInfinity (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if `d` evaluates to NegativeInfinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Examples

The following code example illustrates the use of IsNegativeInfinity:

C#

```
// This will return "true".
Console.WriteLine("IsNegativeInfinity(-5.0 / 0) == {0}.",
Double.IsNegativeInfinity(-5.0 / 0) ? "true" : "false");
```

C#

```
if (d > Double.MaxValue)
   Console.WriteLine("Your number is bigger than a double.");
```

C#

```
// This will equal Infinity.
Console.WriteLine("10.0 minus NegativeInfinity equals {0}.", (10.0 -
Double.NegativeInfinity).ToString());
```

Remarks

Floating-point operations return NegativeInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsInfinity(Double)
- IsPositiveInfinity(Double)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Double.IsNormal(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is normal.

C#

```
public static bool IsNormal (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if the value is normal; `false` otherwise.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsOddInteger(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (double value);
```

Parameters

value Double

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsPositive(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsPositiveInfinity(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to positive infinity.

C#

```
public static bool IsPositiveInfinity (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if `d` evaluates to PositiveInfinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Examples

The following code example illustrates the use of IsPositiveInfinity:

C#

```
// This will return "true".
Console.WriteLine("IsPositiveInfinity(4.0 / 0) == {0}.",
Double.IsPositiveInfinity(4.0 / 0) ? "true" : "false");
```

Remarks

Floating-point operations return PositiveInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsInfinity(Double)
- IsNegativeInfinity(Double)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IsPow2(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsRealNumber(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
public static bool IsRealNumber (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Remarks

This function returns `true` for a complex number `a + bi` where `b` is zero.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IsSubnormal(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is subnormal.

C#

```
public static bool IsSubnormal (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Boolean

`true` if the value is subnormal; `false` otherwise.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Lerp(Double, Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs a linear interpolation between two values based on the given weight.

C#

```
public static double Lerp (double value1, double value2, double amount);
```

Parameters

`value1` Double

The first value, which is intended to be the lower bound.

`value2` Double

The second value, which is intended to be the upper bound.

`amount` Double

A value, intended to be between 0 and 1, that indicates the weight of the interpolation.

Returns

Double

The interpolated value.

Implements

Lerp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Log Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Log(Double, Double)	Computes the logarithm of a value in the specified base.
Log(Double)	Computes the natural (`base-E` logarithm of a value.

Log(Double, Double)

Computes the logarithm of a value in the specified base.

C#

```
public static double Log (double x, double newBase);
```

Parameters

x Double

The value whose logarithm is to be computed.

newBase Double

The base in which the logarithm is to be computed.

Returns

Double

The base-`newBase` logarithm of `x`.

Implements

Log(TSelf, TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Log(Double)

Computes the natural (`base-E` logarithm of a value.

```
C#
public static double Log (double x);
```

Parameters

x Double

The value whose natural logarithm is to be computed.

Returns

Double

The natural logarithm of `x`.

Implements

Log(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Log10(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value.

C#

```
public static double Log10 (double x);
```

Parameters

x Double

The value whose base-10 logarithm is to be computed.

Returns

Double

The base-10 logarithm of `x`.

Implements

Log10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Log10P1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value plus one.

C#

```
public static double Log10P1 (double x);
```

Parameters

x Double

The value to which one is added before computing the base-10 logarithm.

Returns

Double

$\log_{10}(x + 1)$

Implements

Log10P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Log2(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

```
C#
public static double Log2 (double value);
```

Parameters

`value` Double

The value whose log2 is to be computed.

Returns

Double

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Log2P1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-2 logarithm of a value plus one.

C#

```
public static double Log2P1 (double x);
```

Parameters

x Double

The value to which one is added before computing the base-2 logarithm.

Returns

Double

$\log_2(x + 1)$

Implements

Log2P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.LogP1(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the natural (`base-E`) logarithm of a value plus one.

```
C#
public static double LogP1 (double x);
```

Parameters

x Double

The value to which one is added before computing the natural logarithm.

Returns

Double

$\log_e(x + 1)$

Implements

LogP1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Max(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static double Max (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.MaxMagnitude(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static double MaxMagnitude (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf>, this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.MaxMagnitudeNumber(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#
public static double MaxMagnitudeNumber (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.MaxNumber(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
public static double MaxNumber (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0`

to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Min(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static double Min (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf>, this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.MinMagnitude(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static double MinMagnitude (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf>, this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.MinMagnitudeNumber(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#
public static double MinMagnitudeNumber (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.MinNumber(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
public static double MinNumber (double x, double y);
```

Parameters

x Double

The value to compare with `y`.

y Double

The value to compare with `x`.

Returns

Double

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0`

to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its double-precision floating-point number equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its double-precision floating-point number equivalent.
Parse(String)	Converts the string representation of a number to its double-precision floating-point number equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its double-precision floating-point number equivalent.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.

```
C#

public static double Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Double

A double-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a numeric value.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example illustrates the use of the Parse(String, NumberStyles, IFormatProvider) method to assign several string representations of temperature values to a `Temperature` object.

C#

```
using System;
using System.Globalization;

public class Temperature
{
   // Parses the temperature from a string. Temperature scale is
   // indicated by 'F (for Fahrenheit) or 'C (for Celsius) at the end
   // of the string.
   public static Temperature Parse(string s, NumberStyles styles,
                                   IFormatProvider provider)
   {
      Temperature temp = new Temperature();

      if (s.TrimEnd(null).EndsWith("'F"))
      {
         temp.Value = Double.Parse(s.Remove(s.LastIndexOf((char)39), 2),
                                   styles, provider);
      }
      else
      {
```

```
         if (s.TrimEnd(null).EndsWith("'C"))
            temp.Celsius = Double.Parse(s.Remove(s.LastIndexOf((char)39),
2),
                                        styles, provider);
         else
            temp.Value = Double.Parse(s, styles, provider);
      }
      return temp;
   }

   // Declare private constructor so Temperature so only Parse method can
   // create a new instance
   private Temperature()   {}

   protected double m_value;

   public double Value
   {
      get { return m_value; }
      private set { m_value = value; }
   }

   public double Celsius
   {
      get { return (m_value - 32) / 1.8; }
      private set { m_value = value * 1.8 + 32; }
   }

   public double Fahrenheit
   {
      get {return m_value; }
   }
}

public class TestTemperature
{
   public static void Main()
   {
      string value;
      NumberStyles styles;
      IFormatProvider provider;
      Temperature temp;

      value = "25,3'C";
      styles = NumberStyles.Float;
      provider = CultureInfo.CreateSpecificCulture("fr-FR");
      temp = Temperature.Parse(value, styles, provider);
      Console.WriteLine("{0} degrees Fahrenheit equals {1} degrees
Celsius.",
                        temp.Fahrenheit, temp.Celsius);

      value = " (40) 'C";
      styles = NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite
               | NumberStyles.AllowParentheses;
```

```
      provider = NumberFormatInfo.InvariantInfo;
      temp = Temperature.Parse(value, styles, provider);
      Console.WriteLine("{0} degrees Fahrenheit equals {1} degrees
Celsius.",
                        temp.Fahrenheit, temp.Celsius);

      value = "5,778E03'C";      // Approximate surface temperature of
the Sun
      styles = NumberStyles.AllowDecimalPoint |
NumberStyles.AllowThousands |
               NumberStyles.AllowExponent;
      provider = CultureInfo.CreateSpecificCulture("en-GB");
      temp = Temperature.Parse(value, styles, provider);
      Console.WriteLine("{0} degrees Fahrenheit equals {1} degrees
Celsius.",
                        temp.Fahrenheit.ToString("N"),
temp.Celsius.ToString("N"));
   }
}
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `style` parameter defines the style elements (such as white space, thousands separators, and currency symbols) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- AllowHexSpecifier
- HexNumber

The `s` parameter can contain NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol for the culture specified by `provider`. Depending on the value of `style`, it can also take the form:

[*ws*] [*$*] [*sign*][*integral-digits*,]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Elements framed in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	A negative sign symbol (-) or a positive sign symbol (+). The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
,	A culture-specific group separator. The current culture's group separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

Note

Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Double type. The remaining System.Globalization.NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in `s` in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation. This flag by itself supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (,) and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that supplies culture-specific information used in interpreting the format of `s`. Typically, it is a NumberFormatInfo

or CultureInfo object. If `provider` is `null` or a NumberFormatInfo cannot be obtained, the formatting information for the current system culture is used.

Ordinarily, if you pass the Double.Parse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, because of a loss of precision, the values may not be equal. In addition, attempting to parse the string representation of either MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. On .NET Core 3.0 and later versions, it returns Double.NegativeInfinity if you attempt to parse MinValue or Double.PositiveInfinity if you attempt to parse MaxValue. The following example provides an illustration.

C#

```
   string value;

   value = Double.MinValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   value = Double.MaxValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   // Format without the default precision.
   value = Double.MinValue.ToString("G17");
   try
   {
       Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException)
   {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the Double type.
//    1.79769313486232E+308 is outside the range of the Double type.
//    -1.79769313486232E+308
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the Parse(String, NumberStyles, IFormatProvider) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the Parse(String, NumberStyles, IFormatProvider) method will return Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum values of Double than to positive or negative infinity. In those cases, the method returns Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent.

```C#
public static double Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A character span that contains the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Double

A double-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Exceptions

FormatException

`s` does not represent a numeric value.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

If `s` is out of range of the Double data type, the method returns Double.NegativeInfinity if `s` is less than Double.MinValue and Double.PositiveInfinity if `s` is greater than Double.MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static double Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Double

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its double-precision floating-point number equivalent.

C#

```
public static double Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Double

A double-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to a Double value.

C#

```
protected void OkToDouble_Click(object sender, EventArgs e)
{
    string locale;
    double number;
    CultureInfo culture;
```

```
    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Double.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (OverflowException)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

This overload of the Parse(String, IFormatProvider) method is typically used to convert text that can be formatted in a variety of ways to a Double value. For example, it can be used to convert the text that is entered by a user into an HTML text box to a numeric value.

The `s` parameter is interpreted using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. The `s` parameter can contain NumberFormatInfo.PositiveInfinitySymbol,

NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol for the culture specified by `provider`, or it can contain a string of the form:

[*ws*][*sign*]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Optional elements are framed in square brackets ([and]). Elements that contain the term "digits" consist of a series of numeric characters ranging from 0 to 9.

Expand table

Element	Description
ws	A series of white-space characters.
sign	A negative sign symbol (-) or a positive sign symbol (+).
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Runs of *integral-digits* can be partitioned by a group-separator symbol. For example, in some cultures a comma (,) separates groups of thousands. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
.	A culture-specific decimal point symbol.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

For more information about numeric formats, see the Formatting Types topic.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that supplies culture-specific information used in interpreting the format of `s`. Typically, it is a NumberFormatInfo or CultureInfo object. If `provider` is `null` or a NumberFormatInfo cannot be obtained, the formatting information for the current system culture is used.

Ordinarily, if you pass the Double.Parse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, because of a loss of precision, the values may not be equal. In addition, attempting to parse the string representation of either Double.MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. On .NET Core 3.0 and later versions, it returns

Double.NegativeInfinity if you attempt to parse MinValue or Double.PositiveInfinity if you attempt to parse MaxValue. The following example provides an illustration.

C#

```
   string value;

   value = Double.MinValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   value = Double.MaxValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   // Format without the default precision.
   value = Double.MinValue.ToString("G17");
   try
   {
       Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException)
   {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the Double type.
//    1.79769313486232E+308 is outside the range of the Double type.
//    -1.79769313486232E+308
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the Parse(String, IFormatProvider) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the Parse(String, IFormatProvider) method will return Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum

values of Double than to positive or negative infinity. In those cases, the method returns Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

Converts the string representation of a number to its double-precision floating-point number equivalent.

C#

```
public static double Parse (string s);
```

Parameters

`s` String

A string that contains a number to convert.

Returns

Double

A double-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

Examples

The following example illustrates the use of the Parse(String) method.

C#

```
public class Temperature {
    // Parses the temperature from a string in form
    // [ws][sign]digits['F|'C][ws]
    public static Temperature Parse(string s) {
        Temperature temp = new Temperature();

        if( s.TrimEnd(null).EndsWith("'F") ) {
            temp.Value = Double.Parse( s.Remove(s.LastIndexOf('\''), 2)
);
        }
        else if( s.TrimEnd(null).EndsWith("'C") ) {
            temp.Celsius = Double.Parse( s.Remove(s.LastIndexOf('\''), 2)
);
        }
        else {
            temp.Value = Double.Parse(s);
        }
```

```
        return temp;
    }

    // The value holder
    protected double m_value;

    public double Value {
        get {
            return m_value;
        }
        set {
            m_value = value;
        }
    }

    public double Celsius {
        get {
            return (m_value-32.0)/1.8;
        }
        set {
            m_value = 1.8*value+32.0;
        }
    }
}
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `s` parameter can contain the current culture's NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, NumberFormatInfo.NaNSymbol, or a string of the form:

[*ws*][*sign*][*integral-digits*[,]]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters.

Element	Description
sign	A negative sign symbol (-) or a positive sign symbol (+). Only a leading sign can be used.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Runs of *integral-digits* can be partitioned by a group-separator symbol. For example, in some cultures a comma (,) separates groups of thousands. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
,	A culture-specific thousands separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

The `s` parameter is interpreted using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. This means that white space and thousands separators are allowed, for example, while currency symbols are not. For finer control over which style elements are permitted in `s` for the parse operation to succeed, call the Double.Parse(String, NumberStyles) or the Double.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is interpreted using the formatting information in a NumberFormatInfo object that is initialized for the current culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other culture, call the Double.Parse(String, IFormatProvider) or Double.Parse(String, NumberStyles, IFormatProvider) method.

Ordinarily, if you pass the Double.Parse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, on .NET Framework and on .NET Core 2.2 and earlier versions, the values may not be equal because of a loss of precision. In addition, attempting to parse the string representation of either Double.MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. The following example provides an illustration.

```
C#
```

```
   string value;

   value = Double.MinValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   value = Double.MaxValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   // Format without the default precision.
   value = Double.MinValue.ToString("G17");
   try
   {
       Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException)
   {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the Double type.
//    1.79769313486232E+308 is outside the range of the Double type.
//    -1.79769313486232E+308
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the Parse(String) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the method will return Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum values of Double than to positive or negative infinity. In those cases, the method returns Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group

separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- TryParse(String, Double)
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static double Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Double

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static double Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Double

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its double-precision floating-point number equivalent.

```
C#

public static double Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is a combination of Float combined with AllowThousands.

Returns

Double

A double-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Double.MinValue or greater than Double.MaxValue.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` includes the AllowHexSpecifier value.

Examples

The following example uses the Parse(String, NumberStyles) method to parse the string representations of Double values using the en-US culture.

C#

```
public static void Main()
{
   // Set current thread culture to en-US.
   Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("en-US");

   string value;
   NumberStyles styles;

   // Parse a string in exponential notation with only the AllowExponent
flag.
   value = "-1.063E-02";
   styles = NumberStyles.AllowExponent;
   ShowNumericValue(value, styles);

   // Parse a string in exponential notation
   // with the AllowExponent and Number flags.
   styles = NumberStyles.AllowExponent | NumberStyles.Number;
   ShowNumericValue(value, styles);

   // Parse a currency value with leading and trailing white space, and
   // white space after the U.S. currency symbol.
   value = " $ 6,164.3299  ";
   styles = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
   ShowNumericValue(value, styles);

   // Parse negative value with thousands separator and decimal.
```

```
   value = "(4,320.64)";
   styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
            NumberStyles.Float;
   ShowNumericValue(value, styles);

   styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
            NumberStyles.Float | NumberStyles.AllowThousands;
   ShowNumericValue(value, styles);
}

private static void ShowNumericValue(string value, NumberStyles styles)
{
   double number;
   try
   {
      number = Double.Parse(value, styles);
      Console.WriteLine("Converted '{0}' using {1} to {2}.",
                        value, styles.ToString(), number);
   }
   catch (FormatException)
   {
      Console.WriteLine("Unable to parse '{0}' with styles {1}.",
                        value, styles.ToString());
   }
   Console.WriteLine();
}
// The example displays the following output to the console:
//    Unable to parse '-1.063E-02' with styles AllowExponent.
//
//    Converted '-1.063E-02' using AllowTrailingSign, AllowThousands,
Float to -0.01063.
//
//    Converted ' $ 6,164.3299  ' using Number, AllowCurrencySymbol to
6164.3299.
//
//    Unable to parse '(4,320.64)' with styles AllowTrailingSign,
AllowParentheses, Float.
//
//    Converted '(4,320.64)' using AllowTrailingSign, AllowParentheses,
AllowThousands, Float to -4320.64.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `style` parameter defines the style elements (such as white space, thousands separators, and currency symbols) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- AllowHexSpecifier
- HexNumber

The `s` parameter can contain the current culture's NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol. Depending on the value of `style`, it can also take the form:

[*ws*][*$*][*sign*][*integral-digits*[,]]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	A negative sign symbol (-) or a positive sign symbol (+). The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
,	A culture-specific group separator. The current culture's group separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands

Element	Description
	flag
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Double type. The remaining System.Globalization.NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in `s` in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation. This flag by itself supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.

NumberStyles value	Elements permitted in `s` in addition to digits
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (,) and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see CurrentInfo.

Ordinarily, if you pass the Double.Parse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, because of a loss of precision, the values may not be equal. In addition, attempting to parse the string representation of either Double.MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. On .NET Core 3.0 and later versions, it returns Double.NegativeInfinity if you attempt to parse MinValue or Double.PositiveInfinity if you attempt to parse MaxValue. The following example provides an illustration.

C#

```
   string value;

   value = Double.MinValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }
```

```
   value = Double.MaxValue.ToString();
   try {
      Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException) {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }

   // Format without the default precision.
   value = Double.MinValue.ToString("G17");
   try
   {
       Console.WriteLine(Double.Parse(value));
   }
   catch (OverflowException)
   {
      Console.WriteLine($"{value} is outside the range of the Double
type.");
   }
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the Double type.
//    1.79769313486232E+308 is outside the range of the Double type.
//    -1.79769313486232E+308
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the Parse(String, NumberStyles) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the Parse(String, NumberStyles) method will return Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum values of Double than to positive or negative infinity. In those cases, the method returns Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- TryParse

- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Pow(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes a value raised to a given power.

```
C#
public static double Pow (double x, double y);
```

Parameters

x Double

The value which is raised to the power of `x`.

y Double

The power to which `x` is raised.

Returns

Double

`x` raised to the power of `y`.

Implements

Pow(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.RadiansToDegrees(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from radians to degrees.

C#

```
public static double RadiansToDegrees (double radians);
```

Parameters

`radians` Double

The value to convert to degrees.

Returns

Double

The value of `radians` converted to degrees.

Implements

RadiansToDegrees(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ReciprocalEstimate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal of a value.

C#

```
public static double ReciprocalEstimate (double x);
```

Parameters

x Double

The value whose estimate of the reciprocal is to be computed.

Returns

Double

An estimate of the reciprocal of `x`.

Implements

ReciprocalEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ReciprocalSqrtEstimate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal square root of a value.

C#

```
public static double ReciprocalSqrtEstimate (double x);
```

Parameters

x Double

The value whose estimate of the reciprocal square root is to be computed.

Returns

Double

An estimate of the reciprocal square root of `x`.

Implements

ReciprocalSqrtEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.RootN(Double, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the n-th root of a value.

```
C#
public static double RootN (double x, int n);
```

Parameters

x Double

The value whose `n`-th root is to be computed.

n Int32

The degree of the root to be computed.

Returns

Double

The `n`-th root of `x`.

Implements

RootN(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Round(Double)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Double, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Double, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
Round(Double, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

Round(Double)

Rounds a value to the nearest integer using the default rounding mode (ToEven).

C#

```
public static double Round (double x);
```

Parameters

x Double

The value to round.

Returns

Double

The result of rounding `x` to the nearest integer using the default rounding mode.

Implements

Round(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Double, Int32)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

C#

```
public static double Round (double x, int digits);
```

Parameters

`x` Double

The value to round.

`digits` Int32

The number of fractional digits to which `x` should be rounded.

Returns

Double

The result of rounding `x` to `digits` fractional-digits using the default rounding mode.

Implements

Round(TSelf, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Double, MidpointRounding)

Rounds a value to the nearest integer using the specified rounding mode.

C#

```
public static double Round (double x, MidpointRounding mode);
```

Parameters

`x` Double

The value to round.

`mode` MidpointRounding

The mode under which `x` should be rounded.

Returns

Double

The result of rounding `x` to the nearest integer using `mode`.

Implements

Round(TSelf, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Double, Int32, MidpointRounding)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

```
C#

public static double Round (double x, int digits, MidpointRounding mode);
```

Parameters

`x` Double

The value to round.

`digits` Int32

The number of fractional digits to which `x` should be rounded.

`mode` MidpointRounding

The mode under which `x` should be rounded.

Returns

Double

The result of rounding `x` to `digits` fractional-digits using `mode`.

Implements

Round(TSelf, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ScaleB(Double, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the product of a value and its base-radix raised to the specified power.

C#

```
public static double ScaleB (double x, int n);
```

Parameters

x Double

The value which base-radix raised to the power of `n` multiplies.

n Int32

The value to which base-radix is raised before multipliying `x`.

Returns

Double

The product of `x` and base-radix raised to the power of `n`.

Implements

ScaleB(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Sign(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (double value);
```

Parameters

`value` Double

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Sin(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value.

```
C#
public static double Sin (double x);
```

Parameters

x Double

The value, in radians, whose sine is to be computed.

Returns

Double

The sine of `x`.

Implements

Sin(TSelf)

Remarks

This computes `sin(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.SinCos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value.

```
C#
public static (double Sin, double Cos) SinCos (double x);
```

Parameters

x Double

The value, in radians, whose sine and cosine are to be computed.

Returns

ValueTuple<Double,Double>

The sine and cosine of `x`.

Implements

SinCos(TSelf)

Remarks

This computes `(sin(x), cos(x))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.SinCosPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value.

C#

```
public static (double SinPi, double CosPi) SinCosPi (double x);
```

Parameters

x Double

The value, in radians, whose sine and cosine are to be computed.

Returns

ValueTuple<Double,Double>

The sine and cosine of `x`.

Implements

SinCosPi(TSelf)

Remarks

This computes `(sin(x), cos(x))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Sinh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic sine of a value.

C#

```
public static double Sinh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic sine is to be computed.

Returns

Double

The hyperbolic sine of `x`.

Implements

Sinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.SinPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value that has been multiplied by `pi`.

C#

```
public static double SinPi (double x);
```

Parameters

x Double

The value, in half-revolutions, that is multipled by `pi` before computing its sine.

Returns

Double

The sine of `x` multiplied-by `pi`.

Implements

SinPi(TSelf)

Remarks

This computes `sin(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Sqrt(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the square-root of a value.

C#

```
public static double Sqrt (double x);
```

Parameters

x Double

The value whose square-root is to be computed.

Returns

Double

The square-root of `x`.

Implements

Sqrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Tan(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value.

```
C#
public static double Tan (double x);
```

Parameters

x Double

The value, in radians, whose tangent is to be computed.

Returns

Double

The tangent of `x`.

Implements

Tan(TSelf)

Remarks

This computes `tan(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.Tanh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic tangent of a value.

C#

```
public static double Tanh (double x);
```

Parameters

x Double

The value, in radians, whose hyperbolic tangent is to be computed.

Returns

Double

The hyperbolic tangent of `x`.

Implements

Tanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.TanPi(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value that has been multipled by `pi`.

```
C#
public static double TanPi (double x);
```

Parameters

x Double

The value, in half-revolutions, that is multipled by `pi` before computing its tangent.

Returns

Double

The tangent of `x` multiplied-by `pi`.

Implements

TanPi(TSelf)

Remarks

This computes `tan(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Examples

The following example uses the default Double.ToString() method to display the string representations of a number of Double values.

C#

```
double number;

number = 1.6E20;
// Displays 1.6E+20.
Console.WriteLine(number.ToString());

number = 1.6E2;
// Displays 160.
Console.WriteLine(number.ToString());

number = -3.541;
// Displays -3.541.
Console.WriteLine(number.ToString());

number = -1502345222199E-07;
// Displays -150234.5222199.
Console.WriteLine(number.ToString());

number = -150234522219901995 74E-09;
// Displays -15023452221.9902.
Console.WriteLine(number.ToString());

number = .60344;
// Displays 0.60344.
Console.WriteLine(number.ToString());

number = .000000001;
// Displays 1E-09.
Console.WriteLine(number.ToString());
```

The following example illustrates the use of ToString.

C#

```
bool done = false;
string inp;
do {
   Console.Write("Enter a real number: ");
   inp = Console.ReadLine();
   try {
      d = Double.Parse(inp);
      Console.WriteLine("You entered {0}.", d.ToString());
      done = true;
   }
```

```
    catch (FormatException) {
       Console.WriteLine("You did not enter a number.");
    }
    catch (ArgumentNullException) {
       Console.WriteLine("You did not supply any input.");
    }
    catch (OverflowException) {
        Console.WriteLine("The value you entered, {0}, is out of range.",
inp);
    }
} while (!done);
```

Remarks

The ToString() method formats a Double value in the default ("G", or general) format of the current culture. If you want to specify a different format, precision, or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format or precision	Default (current) culture	ToString(String)
A specific format or precision	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or a string of the form:

[sign]integral-digits[.[fractional-digits]][E[sign]exponential-digits]

Optional elements are framed in square brackets ([and]). Elements that contain the term "digits" consist of a series of numeric characters ranging from 0 to 9. The elements listed in the following table are supported.

Expand table

Element	Description
sign	A negative sign or positive sign symbol.
integral-digits	A series of digits specifying the integral part of the number. Integral-digits can be absent if there are fractional-digits.
'.'	A culture-specific decimal point symbol.

Element	Description
fractional-digits	A series of digits specifying the fractional part of the number.
'E'	An uppercase character 'E', indicating exponential (scientific) notation.
exponential-digits	A series of digits specifying an exponent.

Some examples of the return value are "100", "-123,456,789", "123.45E+6", "500", "3.1416", "600", "-0.123", and "-Infinity".

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

```
C#

public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example displays the string representation of two Double values using CultureInfo objects that represent several different cultures.

```
C#

double value;

value = -16325.62015;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));

value = 16034.125E21;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
```

```
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));
// This example displays the following output to the console:
//       -16325.62015
//       -16325.62015
//       -16325,62015
//       1.6034125E+25
//       1.6034125E+25
//       1,6034125E+25
```

The following example illustrates the use of ToString, taking a String and an IFormatProvider as parameters.

C#

```
public class Temperature : IFormattable {
    // IFormattable.ToString implementation.
    public string ToString(string format, IFormatProvider provider) {
        if( format != null ) {
            if( format.Equals("F") ) {
                return String.Format("{0}'F", this.Value.ToString());
            }
            if( format.Equals("C") ) {
                return String.Format("{0}'C", this.Celsius.ToString());
            }
        }

        return m_value.ToString(format, provider);
    }

    // The value holder
    protected double m_value;

    public double Value {
        get {
            return m_value;
        }
        set {
            m_value = value;
        }
    }

    public double Celsius {
        get {
            return (m_value-32.0)/1.8;
        }
        set {
            m_value = 1.8*value+32.0;
        }
```

```
    }
}
```

Remarks

The ToString(IFormatProvider) method formats a Double value in the default ("G", or general) format of a specified culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current)	ToString()
A specific format or precision	Default (current) culture	ToString(String)
A specific format or precision	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or a string of the form:

[sign]integral-digits[.[fractional-digits]][E[sign]exponential-digits]

Optional elements are framed in square brackets ([and]). Elements that contain the term "digits" consist of a series of numeric characters ranging from 0 to 9. The elements listed in the following table are supported.

Expand table

Element	Description
sign	A negative sign or positive sign symbol.
integral-digits	A series of digits specifying the integral part of the number. Integral-digits can be absent if there are fractional-digits.
'.'	A culture-specific decimal point symbol.
fractional-digits	A series of digits specifying the fractional part of the number.
'*E*'	An uppercase character 'E', indicating exponential (scientific) notation.
exponential-digits	A series of digits specifying an exponent.

Some examples of the return value are "100", "-123,456,789", "123.45E+6", "500", "3.1416", "600", "-0.123", and "-Infinity".

This instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a CultureInfo object or a NumberFormatInfo object. The `provider` parameter supplies culture-specific information used in formatting. If `provider` is `null`, the return value is formatted using the NumberFormatInfo object for the current culture.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid.

Examples

The following example defines a numeric value and formats it as a currency value by using the "C" standard numeric format string and as a numeric value to three decimal places by using the "N" standard numeric format string. The result strings are formatted by using the conventions of the en-US culture. For more information on numeric format strings, see Standard Numeric Format Strings and Custom Numeric Format Strings.

```
C#

using System;

public class Example
{
   public static void Main()
   {
      float number = 1764.3789m;

      // Format as a currency value.
      Console.WriteLine(number.ToString("C"));
```

```
      // Format as a numeric value with 3 decimal places.
      Console.WriteLine(number.ToString("N3"));
   }
}
// The example displays the following output:
//       $1,764.38
//       1,764.379
```

The following example displays several Double values using the supported standard numeric format specifiers together with three custom numeric format strings. One of those custom format strings illustrates how to pad a Single value with leading zeros. In addition, the example uses precision specifiers with each standard format specifier except for "R". The values of the precision specifiers range from 0 to 3. To convert the numeric values to strings, the example uses the formatting conventions of the en-US culture.

C#

```
double[] numbers= {1054.32179, -195489100.8377, 1.0437E21,
                   -1.0573e-05};
string[] specifiers = { "C", "E", "e", "F", "G", "N", "P",
                        "R", "#,000.000", "0.###E-000",
                        "000,000,000,000.00###" };
foreach (double number in numbers)
{
   Console.WriteLine("Formatting of {0}:", number);
   foreach (string specifier in specifiers) {
      Console.WriteLine("   {0,-22} {1}",
                        specifier + ":", number.ToString(specifier));
      // Add precision specifiers from 0 to 3.
      if (specifier.Length == 1 & ! specifier.Equals("R")) {
         for (int precision = 0; precision <= 3; precision++) {
            string pSpecifier = String.Format("{0}{1}", specifier,
precision);
            Console.WriteLine("   {0,-22} {1}",
                              pSpecifier + ":",
number.ToString(pSpecifier));
         }
         Console.WriteLine();
      }
   }
   Console.WriteLine();
}
// The example displays the following output to the console:
//       Formatting of 1054.32179:
//          C:                     $1,054.32
//          C0:                    $1,054
//          C1:                    $1,054.3
//          C2:                    $1,054.32
//          C3:                    $1,054.322
//
```

```
//          E:                      1.054322E+003
//          E0:                     1E+003
//          E1:                     1.1E+003
//          E2:                     1.05E+003
//          E3:                     1.054E+003
//
//          e:                      1.054322e+003
//          e0:                     1e+003
//          e1:                     1.1e+003
//          e2:                     1.05e+003
//          e3:                     1.054e+003
//
//          F:                      1054.32
//          F0:                     1054
//          F1:                     1054.3
//          F2:                     1054.32
//          F3:                     1054.322
//
//          G:                      1054.32179
//          G0:                     1054.32179
//          G1:                     1E+03
//          G2:                     1.1E+03
//          G3:                     1.05E+03
//
//          N:                      1,054.32
//          N0:                     1,054
//          N1:                     1,054.3
//          N2:                     1,054.32
//          N3:                     1,054.322
//
//          P:                      105,432.18 %
//          P0:                     105,432 %
//          P1:                     105,432.2 %
//          P2:                     105,432.18 %
//          P3:                     105,432.179 %
//
//          R:                      1054.32179
//          #,000.000:              1,054.322
//          0.###E-000:             1.054E003
//          000,000,000,000.00###: 000,000,001,054.32179
//
//       Formatting of -195489100.8377:
//          C:                      ($195,489,100.84)
//          C0:                     ($195,489,101)
//          C1:                     ($195,489,100.8)
//          C2:                     ($195,489,100.84)
//          C3:                     ($195,489,100.838)
//
//          E:                      -1.954891E+008
//          E0:                     -2E+008
//          E1:                     -2.0E+008
//          E2:                     -1.95E+008
//          E3:                     -1.955E+008
//
//          e:                      -1.954891e+008
```

```
//          e0:                     -2e+008
//          e1:                     -2.0e+008
//          e2:                     -1.95e+008
//          e3:                     -1.955e+008
//
//          F:                      -195489100.84
//          F0:                     -195489101
//          F1:                     -195489100.8
//          F2:                     -195489100.84
//          F3:                     -195489100.838
//
//          G:                      -195489100.8377
//          G0:                     -195489100.8377
//          G1:                     -2E+08
//          G2:                     -2E+08
//          G3:                     -1.95E+08
//
//          N:                      -195,489,100.84
//          N0:                     -195,489,101
//          N1:                     -195,489,100.8
//          N2:                     -195,489,100.84
//          N3:                     -195,489,100.838
//
//          P:                      -19,548,910,083.77 %
//          P0:                     -19,548,910,084 %
//          P1:                     -19,548,910,083.8 %
//          P2:                     -19,548,910,083.77 %
//          P3:                     -19,548,910,083.770 %
//
//          R:                      -195489100.8377
//          #,000.000:              -195,489,100.838
//          0.###E-000:             -1.955E008
//          000,000,000,000.00###: -000,195,489,100.8377
//
//       Formatting of 1.0437E+21:
//          C:                      $1,043,700,000,000,000,000,000.00
//          C0:                     $1,043,700,000,000,000,000,000
//          C1:                     $1,043,700,000,000,000,000,000.0
//          C2:                     $1,043,700,000,000,000,000,000.00
//          C3:                     $1,043,700,000,000,000,000,000.000
//
//          E:                      1.043700E+021
//          E0:                     1E+021
//          E1:                     1.0E+021
//          E2:                     1.04E+021
//          E3:                     1.044E+021
//
//          e:                      1.043700e+021
//          e0:                     1e+021
//          e1:                     1.0e+021
//          e2:                     1.04e+021
//          e3:                     1.044e+021
//
//          F:                      1043700000000000000000.00
//          F0:                     1043700000000000000000
```

```
//          F1:                     104370000000000000000.0
//          F2:                     104370000000000000000.00
//          F3:                     104370000000000000000.000
//
//          G:                      1.0437E+21
//          G0:                     1.0437E+21
//          G1:                     1E+21
//          G2:                     1E+21
//          G3:                     1.04E+21
//
//          N:                      1,043,700,000,000,000,000,000.00
//          N0:                     1,043,700,000,000,000,000,000
//          N1:                     1,043,700,000,000,000,000,000.0
//          N2:                     1,043,700,000,000,000,000,000.00
//          N3:                     1,043,700,000,000,000,000,000.000
//
//          P:                      104,370,000,000,000,000,000,000.00 %
//          P0:                     104,370,000,000,000,000,000,000 %
//          P1:                     104,370,000,000,000,000,000,000.0 %
//          P2:                     104,370,000,000,000,000,000,000.00 %
//          P3:                     104,370,000,000,000,000,000,000.000 %
//
//          R:                      1.0437E+21
//          #,000.000:              1,043,700,000,000,000,000,000.000
//          0.###E-000:             1.044E021
//          000,000,000,000.00###: 1,043,700,000,000,000,000,000.00
//
//       Formatting of -1.0573E-05:
//          C:                      $0.00
//          C0:                     $0
//          C1:                     $0.0
//          C2:                     $0.00
//          C3:                     $0.000
//
//          E:                      -1.057300E-005
//          E0:                     -1E-005
//          E1:                     -1.1E-005
//          E2:                     -1.06E-005
//          E3:                     -1.057E-005
//
//          e:                      -1.057300e-005
//          e0:                     -1e-005
//          e1:                     -1.1e-005
//          e2:                     -1.06e-005
//          e3:                     -1.057e-005
//
//          F:                      0.00
//          F0:                     0
//          F1:                     0.0
//          F2:                     0.00
//          F3:                     0.000
//
//          G:                      -1.0573E-05
//          G0:                     -1.0573E-05
//          G1:                     -1E-05
```

```
//          G2:                     -1.1E-05
//          G3:                     -1.06E-05
//
//          N:                      0.00
//          N0:                     0
//          N1:                     0.0
//          N2:                     0.00
//          N3:                     0.000
//
//          P:                      0.00 %
//          P0:                     0 %
//          P1:                     0.0 %
//          P2:                     0.00 %
//          P3:                     -0.001 %
//
//          R:                      -1.0573E-05
//          #,000.000:              000.000
//          0.###E-000:             -1.057E-005
//          000,000,000,000.00###: -000,000,000,000.00001
```

Remarks

The ToString(String) method formats a Double value in a specified format by using the conventions of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format or precision	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or the string representation of a number, as specified by `format`.

The `format` parameter can be any valid standard numeric format specifier except for D and X, as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string, the return value is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

By default, the return value only contains 15 digits of precision although a maximum of 17 digits is maintained internally. If the value of this instance has greater than 15 digits, ToString returns PositiveInfinitySymbol or NegativeInfinitySymbol instead of the expected number. If you require more precision, specify `format` with the "G17" format specification, which always returns 17 digits of precision, or "R", which returns 15 digits if the number can be represented with that precision or 17 digits if the number can only be represented with maximum precision.

Notes to Callers

In some cases, Double values formatted with the "R" standard numeric format string do not successfully round-trip if compiled using the `/platform:x64` or `/platform:anycpu` switches and run on 64-bit systems. To work around this problem, you can format Double values by using the "G17" standard numeric format string. The following example uses the "R" format string with a Double value that does not round-trip successfully, and also uses the "G17" format string to successfully round-trip the original value.

C#

```
using System;

public class Example
{
   static void Main(string[] args)
   {
      Console.WriteLine("Attempting to round-trip a Double with 'R':");
      double initialValue = 0.6822871999174;
      string valueString = initialValue.ToString("R");
      double roundTripped = double.Parse(valueString);
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped,
initialValue.Equals(roundTripped));

      Console.WriteLine("Attempting to round-trip a Double with 'G17':");
      string valueString17 = initialValue.ToString("G17");
      double roundTripped17 = double.Parse(valueString17);
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped17,
initialValue.Equals(roundTripped17));
   }
}
```

```
// If compiled to an application that targets anycpu or x64 and run on an
x64 system,
// the example displays the following output:
//       Attempting to round-trip a Double with 'R':
//       0.6822871999174 = 0.68228719991740006: False
//
//       Attempting to round-trip a Double with 'G17':
//       0.6822871999174 = 0.6822871999174: True
```

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Examples

The following example displays a Double value using each of the supported standard numeric format specifiers for several different cultures.

C#

```
double value = 16325.62901;
string specifier;
CultureInfo culture;

// Use standard numeric format specifiers.
specifier = "G";
culture = CultureInfo.CreateSpecificCulture("eu-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,62901
Console.WriteLine(value.ToString(specifier,
CultureInfo.InvariantCulture));
// Displays:    16325.62901

specifier = "C";
culture = CultureInfo.CreateSpecificCulture("en-US");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    $16,325.63
culture = CultureInfo.CreateSpecificCulture("en-GB");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    £16,325.63

specifier = "E04";
culture = CultureInfo.CreateSpecificCulture("sv-SE");
```

```
Console.WriteLine(value.ToString(specifier, culture));
// Displays: 1,6326E+004
 culture = CultureInfo.CreateSpecificCulture("en-NZ");
 Console.WriteLine(value.ToString(specifier, culture));
// Displays:    1.6326E+004

specifier = "F";
culture = CultureInfo.CreateSpecificCulture("fr-FR");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,63
culture = CultureInfo.CreateSpecificCulture("en-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325.63

specifier = "N";
culture = CultureInfo.CreateSpecificCulture("es-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16.325,63
culture = CultureInfo.CreateSpecificCulture("fr-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16 325,63

specifier = "P";
culture = CultureInfo.InvariantCulture;
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.26 %
culture = CultureInfo.CreateSpecificCulture("ar-EG");
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.256 %
```

The following example illustrates the use of ToString, taking a String and an IFormatProvider as parameters.

C#

```
public class Temperature : IFormattable {
    // IFormattable.ToString implementation.
    public string ToString(string format, IFormatProvider provider) {
        if( format != null ) {
            if( format.Equals("F") ) {
                return String.Format("{0}'F", this.Value.ToString());
            }
            if( format.Equals("C") ) {
                return String.Format("{0}'C", this.Celsius.ToString());
            }
        }

        return m_value.ToString(format, provider);
    }

    // The value holder
    protected double m_value;
```

```
    public double Value {
        get {
            return m_value;
        }
        set {
            m_value = value;
        }
    }

    public double Celsius {
        get {
            return (m_value-32.0)/1.8;
        }
        set {
            m_value = 1.8*value+32.0;
        }
    }
}
```

Remarks

The ToString(String, IFormatProvider) method formats a Double value in a specified format of a specified culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format or precision	Default (current) culture	ToString(String)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or the string representation of a number, as specified by `format`.

The `format` parameter can be any valid standard numeric format specifier except for D and X, as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string, the return value for this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a CultureInfo object or a NumberFormatInfo object. The `provider` parameter supplies culture-specific information used in formatting. If `provider` is `null`, the return value is formatted using the NumberFormatInfo object for the current culture.

By default, the return value only contains 15 digits of precision although a maximum of 17 digits is maintained internally. If the value of this instance has greater than 15 digits, ToString returns PositiveInfinitySymbol or NegativeInfinitySymbol instead of the expected number. If you require more precision, specify `format` with the "G17" format specification, which always returns 17 digits of precision, or "R", which returns 15 digits if the number can be represented with that precision or 17 digits if the number can only be represented with maximum precision.

Notes to Callers

In some cases, Double values formatted with the "R" standard numeric format string do not successfully round-trip if compiled using the `/platform:x64` or `/platform:anycpu` switches and run on 64-bit systems. To work around this problem, you can format Double values by using the "G17" standard numeric format string. The following example uses the "R" format string with a Double value that does not round-trip successfully, and also uses the "G17" format string to successfully round-trip the original value.

C#

```
using System;
using System.Globalization;

public class Example
{
   static void Main(string[] args)
   {
      Console.WriteLine("Attempting to round-trip a Double with 'R':");
      double initialValue = 0.6822871999174;
      string valueString = initialValue.ToString("R",

CultureInfo.InvariantCulture);
      double roundTripped = double.Parse(valueString,
                                         CultureInfo.InvariantCulture);
```

```
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped,
initialValue.Equals(roundTripped));

      Console.WriteLine("Attempting to round-trip a Double with 'G17':");
      string valueString17 = initialValue.ToString("G17",

CultureInfo.InvariantCulture);
      double roundTripped17 = double.Parse(valueString17,
                                           CultureInfo.InvariantCulture);
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped17,
initialValue.Equals(roundTripped17));
      // If compiled to an application that targets anycpu or x64 and run
on an x64 system,
      // the example displays the following output:
      //       Attempting to round-trip a Double with 'R':
      //       0.6822871999174 = 0.68228719991740006: False
      //
      //       Attempting to round-trip a Double with 'G17':
      //       0.6822871999174 = 0.6822871999174: True
   }
}
```

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Truncate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Truncates a value.

C#

```
public static double Truncate (double x);
```

Parameters

x Double

The value to truncate.

Returns

Double

The truncation of `x`.

Implements

Truncate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current double instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current double instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

```
C#
```

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, IFormatProvider, Double)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, Double)	Converts the span representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Double)	Converts the string representation of a number to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Double)	Tries to parse a span of UTF-8 characters into a value.
TryParse(String, IFormatProvider, Double)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Double)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Double)	Tries to convert a UTF-8 character span containing the string representation of a number to its double-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider,	Converts a character span containing the string representation of a number in a specified style and

Double)	culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Double)	Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

TryParse(ReadOnlySpan<Char>, IFormatProvider, Double)

Tries to parse a span of characters into a value.

```C#
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out double result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result Double

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, Double)

Converts the span representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, out double result);
```

Parameters

`s` ReadOnlySpan<Char>

A character span that contains the string representation of the number to convert.

`result` Double

When this method returns, contains the double-precision floating-point number equivalent of the numeric value or symbol contained in `s` parameter, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty, or is not in a format compliant with `style`. The conversion also fails if `style` is not a valid combination of NumberStyles enumerated constants. If `s` is a valid number less than Double.MinValue, `result` is NegativeInfinity. If `s` is a valid number greater than Double.MaxValue, `result` is PositiveInfinity. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Double)

Converts the string representation of a number to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, out double result);
```

Parameters

`s` String

A string containing a number to convert.

`result` Double

When this method returns, contains the double-precision floating-point number equivalent of the `s` parameter, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty or is not a number in a valid format. It also fails on .NET Framework and .NET Core 2.2 and earlier versions

if `s` represents a number less than Double.MinValue or greater than Double.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example uses the TryParse(String, Double) method to convert the string representations of numeric values to Double values. It assumes that en-US is the current culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "1,643.57", "$1,643.57", "-1.643e6",
                          "-168934617882109132", "123AE6",
                          null, String.Empty, "ABCDEF" };
      double number;

      foreach (var value in values) {
         if (Double.TryParse(value, out number))
            Console.WriteLine("'{0}' --> {1}", value, number);
         else
            Console.WriteLine("Unable to parse '{0}'.", value);
      }
   }
}
// The example displays the following output:
//       '1,643.57' --> 1643.57
//       Unable to parse '$1,643.57'.
//       '-1.643e6' --> -1643000
//       '-168934617882109132' --> -1.68934617882109E+17
//       Unable to parse '123AE6'.
//       Unable to parse ''.
//       Unable to parse ''.
//       Unable to parse 'ABCDEF'.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

This overload differs from the Double.Parse(String) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed numeric value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `s` parameter can contain the current culture's NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, NumberFormatInfo.NaNSymbol (the string comparison is case-sensitive), or a string of the form:

[ws][sign][integral-digits,]integral-digits[.[fractional-digits]][e[sign]exponential-digits][ws]

Elements in square brackets are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters.
sign	A negative sign or positive sign symbol.
integral-digits	A series of numeric characters ranging from 0 to 9 that specify the integral part of the number. Integral-digits can be absent if there are fractional-digits.
,	A culture-specific group separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A series of numeric characters ranging from 0 to 9 that specify the fractional part of the number.
E	An uppercase or lowercase character 'e', that indicates exponential (scientific) notation.
exponential-digits	A series of numeric characters ranging from 0 to 9 that specify an exponent.

For more information about numeric formats, see Formatting Types.

The `s` parameter is interpreted by using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use the Double.TryParse(String, NumberStyles, IFormatProvider, Double) method overload.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string using the formatting information of some other specified culture, use the Double.TryParse(String, NumberStyles, IFormatProvider, Double) method overload.

Ordinarily, if you pass the Double.TryParse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, because of a loss of precision, the values may not be equal. In addition, attempting to parse the string representation of either Double.MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. On .NET Core 3.0 and later versions, it returns Double.NegativeInfinity if you attempt to parse MinValue or Double.PositiveInfinity if you attempt to parse MaxValue. The following example provides an illustration.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string value;
      double number;

      value = Double.MinValue.ToString();
      if (Double.TryParse(value, out number))
         Console.WriteLine(number);
      else
         Console.WriteLine("{0} is outside the range of a Double.",
                           value);

      value = Double.MaxValue.ToString();
      if (Double.TryParse(value, out number))
         Console.WriteLine(number);
      else
         Console.WriteLine("{0} is outside the range of a Double.",
                           value);
```

```
    }
}
// The example displays the following output:
//    -1.79769313486232E+308 is outside the range of the Double type.
//    1.79769313486232E+308 is outside the range of the Double type.
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the TryParse(String, Double) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the TryParse(String, Double) method calculates a result of Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum values of Double than to positive or negative infinity. In those cases, the method calculates a result of Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Double)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out double result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Double

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, IFormatProvider, Double)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
double result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Double

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Double)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
double result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Double

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Double)

Tries to convert a UTF-8 character span containing the string representation of a number to its double-precision floating-point number equivalent.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text, out double
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A read-only UTF-8 character span that contains the number to convert.

`result` Double

When this method returns, contains a double-precision floating-point number equivalent of the numeric value or symbol contained in `utf8Text` if the conversion succeeded or zero if the conversion failed. The conversion fails if the `utf8Text` is Empty or is not in a valid format. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Double)

Converts a character span containing the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
```

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
double result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only character span that contains the number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the permitted format of `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Double

When this method returns and if the conversion succeeded, contains a double-precision floating-point number equivalent of the numeric value or symbol contained in `s`. Contains zero if the conversion failed. The conversion fails if the `s` parameter is `null`, an empty character span, or not a number in a format compliant with `style`. If `s` is a valid number less than Double.MinValue, `result` is NegativeInfinity. If `s` is a valid number greater than Double.MaxValue, `result` is PositiveInfinity. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Double)

Converts the string representation of a number in a specified style and culture-specific format to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out double result);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the permitted format of `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

`result` Double

When this method returns, contains a double-precision floating-point number equivalent of the numeric value or symbol contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty or is not in a format compliant with `style`, or if `style` is not a valid combination of NumberStyles enumeration constants. It also fails on .NET

Framework or .NET Core 2.2 and earlier versions if `s` represents a number less than SByte.MinValue or greater than SByte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` includes the AllowHexSpecifier value.

Examples

The following example demonstrates the use of the Double.TryParse(String, NumberStyles, IFormatProvider, Double) method to parse the string representation of numbers that have a particular style and are formatted using the conventions of a particular culture.

C#

```
string value;
NumberStyles style;
CultureInfo culture;
double number;

// Parse currency value using en-GB culture.
value = "£1,097.63";
style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
culture = CultureInfo.CreateSpecificCulture("en-GB");
if (Double.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '£1,097.63' to 1097.63.

value = "1345,978";
style = NumberStyles.AllowDecimalPoint;
culture = CultureInfo.CreateSpecificCulture("fr-FR");
```

```
if (Double.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '1345,978' to 1345.978.

value = "1.345,978";
style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowThousands;
culture = CultureInfo.CreateSpecificCulture("es-ES");
if (Double.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Converted '1.345,978' to 1345.978.

value = "1 345,978";
if (Double.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// Displays:
//       Unable to convert '1 345,978'.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The TryParse method is like the Parse(String, NumberStyles, IFormatProvider) method, except this method does not throw an exception if the conversion fails. If the conversion succeeds, the return value is `true` and the `result` parameter is set to the outcome of the conversion. If the conversion fails, the return value is `false` and the `result` parameter is set to zero. This eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `style` parameter defines the allowable format of the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- NumberStyles.AllowHexSpecifier
- NumberStyles.HexNumber

The `s` parameter can contain NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol for the culture indicated by `provider`. In addition, depending on the value of `style`, the `s` parameter may include the following elements:

[ws] [$] [sign][integral-digits,]integral-digits[.fractional-digits][e[sign]exponential-digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag. It can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern or NumberFormatInfo.CurrencyPositivePattern properties of the NumberFormatInfo object returned by the IFormatProvider.GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Integral-digits can be absent if there are fractional-digits.
,	A culture-specific thousands separator symbol. The current culture's thousands separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.

Element	Description
e	The e or E character, which indicates that `s` can represent a number using exponential notation. The `s` parameter can represent a number in exponential notation if style includes the NumberStyles.AllowExponent flag.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Double type. The remaining System.Globalization.NumberStyles members control elements that may be but are not required to be present in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The . and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation. This flag by itself supports values in the form *integral-digits*E*exponential-digits*; additional flags are needed to successfully parse strings in exponential notation with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The , element.

NumberStyles value	Elements permitted in s in addition to digits
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the . symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (*,*), and decimal point (.) elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

The `provider` parameter is a IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null` or a NumberFormatInfo object cannot be obtained, the format information for the current culture is used.

The conversion fails if the `s` parameter is `null` or not a numeric value, the `provider` parameter does not yield a NumberFormatInfo object, or the `style` parameter is not a combination of bit flags from the NumberStyles enumeration.

Ordinarily, if you pass the Double.TryParse method a string that is created by calling the Double.ToString method, the original Double value is returned. However, because of a loss of precision, the values may not be equal. In addition, attempting to parse the string representation of either Double.MinValue or Double.MaxValue fails to round-trip. On .NET Framework and .NET Core 2.2 and previous versions, it throws an OverflowException. On .NET Core 3.0 and later versions, it returns Double.NegativeInfinity if you attempt to parse MinValue or Double.PositiveInfinity if you attempt to parse MaxValue. The following example provides an illustration.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string value;
      double number;

      value = Double.MinValue.ToString();
      if (Double.TryParse(value, out number))
         Console.WriteLine(number);
```

```
        else
           Console.WriteLine("{0} is outside the range of a Double.",
                             value);

        value = Double.MaxValue.ToString();
        if (Double.TryParse(value, out number))
           Console.WriteLine(number);
        else
           Console.WriteLine("{0} is outside the range of a Double.",
                             value);
     }
  }
  // The example displays the following output:
  //    -1.79769313486232E+308 is outside the range of the Double type.
  //    1.79769313486232E+308 is outside the range of the Double type.
```

On .NET Framework and .NET Core 2.2 and earlier versions, if `s` is out of range of the Double data type, the Double.TryParse(String, NumberStyles, IFormatProvider, Double) method throws an OverflowException.

On .NET Core 3.0 and later versions, no exception is thrown when `s` is out of range of the Double data type. In most cases, the Double.TryParse(String, NumberStyles, IFormatProvider, Double) method calculates a result of Double.PositiveInfinity or Double.NegativeInfinity. However, there is a small set of values that are considered to be closer to the maximum or minimum values of Double than to positive or negative infinity. In those cases, the method calculates a result of Double.MaxValue or Double.MinValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Equality(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Double values are equal.

```
C#
public static bool operator == (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are equal; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Remarks

The Equality method defines the equality operator for Double values.

If two Double.NaN values are tested for equality by using the equality operator, the result is `false`; two Double.NaN values are not considered equal. If they are tested for

equality by calling the Equals method, the result is `true`. When you want to determine whether the value of a Double is not a number (NaN), an alternative is to call the IsNaN method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.GreaterThan(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Double value is greater than another specified Double value.

```
C#
public static bool operator > (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Remarks

The GreaterThan method defines the operation of the greater-than operator for Double values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.GreaterThanOrEqual(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Double value is greater than or equal to another specified Double value.

```
C#

public static bool operator >= (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Remarks

The GreaterThanOrEqual method defines the operation of the greater-than-or-equal operator for Double values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.Inequality(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Double values are not equal.

```
C#
public static bool operator != (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are not equal; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Remarks

The Inequality method defines the inequality operator for Double values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.LessThan(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Double value is less than another specified Double value.

```
C#
public static bool operator < (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Remarks

The LessThan method defines the operation of the less-than operator for Double values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.LessThanOrEqual(Double, Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Double value is less than or equal to another specified Double value.

```
C#
public static bool operator <= (double left, double right);
```

Parameters

`left` Double

The first value to compare.

`right` Double

The second value to compare.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Remarks

The LessThanOrEqual method defines the operation of the less-than-or-equal operator for Double values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

This conversion is not supported. No value is returned.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, unchanged.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToSByte(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this Double value.

provider IFormatProvider

An IFormatProvider implementation that supplies culture-specific information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt16(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt32(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Double instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt64(Double) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.IAdditionOperators<Double,Double,Double>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static double IAdditionOperators<double,double,double>.operator + (double
left, double right);
```

Parameters

`left` Double

The value to which `right` is added.

`right` Double

The value that's added to `left`.

Returns

Double

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IAdditiveIdentity<Double,Double>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static double
System.Numerics.IAdditiveIdentity<System.Double,System.Double>.AdditiveIdentity { get; }
```

Property Value

Double

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IBinaryNumber<Double>.AllBits Set Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static double System.Numerics.IBinaryNumber<System.Double>.AllBitsSet { get;
}
```

Property Value

Double

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IBitwiseOperators<Double,Double,Double>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static double IBitwiseOperators<double,double,double>.operator & (double
left, double right);
```

Parameters

`left` Double

The value to `and` with `right`.

`right` Double

The value to `and` with `left`.

Returns

Double

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IBitwiseOperators<Double,Double,Double>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static double IBitwiseOperators<double,double,double>.operator | (double
left, double right);
```

Parameters

`left` Double

The value to `or` with `right`.

`right` Double

The value to `or` with `left`.

Returns

Double

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IBitwiseOperators<Double,Double,Double>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static double IBitwiseOperators<double,double,double>.operator ^ (double
left, double right);
```

Parameters

`left` Double

The value to xor with `right`.

`right` Double

The value to xor with `left`.

Returns

Double

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IBitwise Operators<Double,Double,Double>.OnesComplement Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#

static double IBitwiseOperators<double,double,double>.operator ~ (double
value);
```

Parameters

`value` Double

The value for which to compute the ones-complement.

Returns

Double

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IDecrementOperators<Double>.Decrement(Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static double IDecrementOperators<double>.operator -- (double value);
```

Parameters

`value` Double

The value to decrement.

Returns

Double

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IDivisionOperators<Double,Double,Double>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static double IDivisionOperators<double,double,double>.operator / (double
left, double right);
```

Parameters

`left` Double

The value that `right` divides.

`right` Double

The value that divides `left`.

Returns

Double

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IFloatingPoint<Double>.GetExponentByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<double>.GetExponentByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

Implements

GetExponentByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IFloatingPoint<Double>.GetExponentShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current exponent.

C#

```
int IFloatingPoint<double>.GetExponentShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current exponent.

Implements

GetExponentShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IFloatingPoint<Double>.GetSignificandBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the current significand.

C#

```
int IFloatingPoint<double>.GetSignificandBitLength ();
```

Returns

Int32

The length, in bits, of the current significand.

Implements

GetSignificandBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Double.IFloatingPoint<Double>.GetSignificandByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<double>.GetSignificandByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

Implements

GetSignificandByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IFloatingPoint<Double>.TryWriteExponentBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in big-endian format, to a given span.

```
C#

bool IFloatingPoint<double>.TryWriteExponentBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IFloatingPoint<Double>.TryWriteExponentLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<double>.TryWriteExponentLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IFloatingPoint<Double>.TryWriteSignificandBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<double>.TryWriteSignificandBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IFloatingPoint<Double>.TryWriteSignificandLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<double>.TryWriteSignificandLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IFloatingPoint Constants<Double>.E Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `e`.

```
C#

static double System.Numerics.IFloatingPointConstants<System.Double>.E {
get; }
```

Property Value

Double

Implements

E

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IFloatingPoint Constants<Double>.Pi Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `pi`.

C#

```
static double System.Numerics.IFloatingPointConstants<System.Double>.Pi {
get; }
```

Property Value

Double

Implements

Pi

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IFloatingPoint Constants<Double>.Tau Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `tau`.

C#

```
static double System.Numerics.IFloatingPointConstants<System.Double>.Tau {
get; }
```

Property Value

Double

Implements

Tau

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IFloatingPoint Ieee754<Double>.Epsilon Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the smallest value such that can be added to `0` that does not result in `0`.

```
C#

static double System.Numerics.IFloatingPointIeee754<System.Double>.Epsilon {
get; }
```

Property Value

Double

Implements

Epsilon

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IFloatingPoint Ieee754<Double>.NaN Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents `NaN`.

C#

```
static double System.Numerics.IFloatingPointIeee754<System.Double>.NaN {
get; }
```

Property Value

Double

Implements

NaN

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.IFloatingPointIeee754<Double>.NegativeInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents negative `infinity`.

C#

```
static double
System.Numerics.IFloatingPointIeee754<System.Double>.NegativeInfinity { get;
}
```

Property Value

Double

Implements

NegativeInfinity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IFloatingPoint Ieee754<Double>.NegativeZero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents negative `zero`.

C#

```
static double
System.Numerics.IFloatingPointIeee754<System.Double>.NegativeZero { get; }
```

Property Value

Double

Implements

NegativeZero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Double.IFloatingPointIeee754<Double>.PositiveInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents positive `infinity`.

C#

```
static double
System.Numerics.IFloatingPointIeee754<System.Double>.PositiveInfinity { get;
}
```

Property Value

Double

Implements

PositiveInfinity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IIncrementOperators<Double>.Increment(Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static double IIncrementOperators<double>.operator ++ (double value);
```

Parameters

`value` Double

The value to increment.

Returns

Double

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IMinMaxValue<Double>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static double System.Numerics.IMinMaxValue<System.Double>.MaxValue { get; }
```

Property Value

Double

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IMinMaxValue<Double>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static double System.Numerics.IMinMaxValue<System.Double>.MinValue { get; }
```

Property Value

Double

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IModulus Operators<Double,Double,Double>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static double IModulusOperators<double,double,double>.operator % (double
left, double right);
```

Parameters

`left` Double

The value that `right` divides.

`right` Double

The value that divides `left`.

Returns

Double

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IMultiplicativeIdentity<Double,Double>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static double
System.Numerics.IMultiplicativeIdentity<System.Double,System.Double>.MultiplicativeIdentity { get; }
```

Property Value

Double

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.IMultiplyOperators<Double,Double,Double>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static double IMultiplyOperators<double,double,double>.operator * (double
left, double right);
```

Parameters

`left` Double

The value that `right` multiplies.

`right` Double

The value that multiplies `left`.

Returns

Double

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.IsCanonical(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<double>.IsCanonical (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.IsComplexNumber(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<double>.IsComplexNumber (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.IsImaginaryNumber(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<double>.IsImaginaryNumber (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.IsZero(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<double>.IsZero (double value);
```

Parameters

`value` Double

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static double System.Numerics.INumberBase<System.Double>.One { get; }
```

Property Value

Double

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.INumberBase<Double>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Double>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<double>.TryConvertFromChecked<TOther> (TOther value,
out double result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Double

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Double.INumberBase<Double>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<double>.TryConvertFromSaturating<TOther> (TOther
value, out double result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Double

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<double>.TryConvertFromTruncating<TOther> (TOther
value, out double result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Double

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<double>.TryConvertToChecked<TOther> (double value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Double

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Double.INumberBase<Double>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<double>.TryConvertToSaturating<TOther> (double value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Double

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<double>.TryConvertToTruncating<TOther> (double value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Double

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.INumberBase<Double>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static double System.Numerics.INumberBase<System.Double>.Zero { get; }
```

Property Value

Double

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Double.ISignedNumber<Double>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
static double System.Numerics.ISignedNumber<System.Double>.NegativeOne {
get; }
```

Property Value

Double

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Double.ISubtraction Operators<Double,Double,Double>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static double ISubtractionOperators<double,double,double>.operator - (double
left, double right);
```

Parameters

`left` Double

The value from which `right` is subtracted.

`right` Double

The value that's subtracted from `left`.

Returns

Double

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IUnaryNegationOperators<Double,Double>.UnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#

static double IUnaryNegationOperators<double,double>.operator - (double
value);
```

Parameters

`value` Double

The value for which to compute the unary negation.

Returns

Double

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Double.IUnaryPlusOperators<Double,Double>.UnaryPlus(Double) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static double IUnaryPlusOperators<double,double>.operator + (double value);
```

Parameters

`value` Double

The value for which to compute the unary plus.

Returns

Double

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

DuplicateWaitObjectException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an object appears more than once in an array of synchronization objects.

C#

```
public class DuplicateWaitObjectException : ArgumentException
```

Inheritance Object → Exception → SystemException → ArgumentException → DuplicateWaitObjectException

Remarks

The common language runtime provides a thread synchronization mechanism based on synchronization objects waiting for execution in an array of WaitHandle objects. If the array of WaitHandle objects passed to WaitAll or WaitAny contains any duplicate operating system handles, DuplicateWaitObjectException is thrown. For more information, see WaitHandle.

DuplicateWaitObjectException uses the HRESULT COR_E_DUPLICATEWAITOBJECT, which has the value 0x80131529.

For a list of initial property values for an instance of DuplicateWaitObjectException, see the DuplicateWaitObjectException constructors.

> **Note**
>
> This exception is not included in **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **ArgumentException** instead.

Constructors

Expand table

DuplicateWaitObjectException()	Initializes a new instance of the DuplicateWaitObjectException class.
DuplicateWaitObject Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DuplicateWaitObjectException class with serialized data.
DuplicateWaitObject Exception(String)	Initializes a new instance of the DuplicateWaitObjectException class with the name of the parameter that causes this exception.
DuplicateWaitObject Exception(String, Exception)	Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and a reference to the inner exception that is the cause of this exception.
DuplicateWaitObject Exception(String, String)	Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and the name of the parameter that causes this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message and the parameter name, or only the error message if no parameter name is set. (Inherited from ArgumentException)

ParamName	Gets the name of the parameter that causes this exception. (Inherited from ArgumentException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the parameter name and additional exception information. (Inherited from ArgumentException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- WaitHandle
- Exception
- Managed Threading
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DuplicateWaitObjectException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the DuplicateWaitObjectException class.

Overloads

Expand table

DuplicateWaitObjectException()	Initializes a new instance of the DuplicateWaitObjectException class.
DuplicateWaitObjectException(String)	Initializes a new instance of the DuplicateWaitObjectException class with the name of the parameter that causes this exception.
DuplicateWaitObjectException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the DuplicateWaitObjectException class with serialized data.
DuplicateWaitObjectException(String, Exception)	Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and a reference to the inner exception that is the cause of this exception.
DuplicateWaitObjectException(String, String)	Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and the name of the parameter that causes this exception.

DuplicateWaitObjectException()

Initializes a new instance of the DuplicateWaitObjectException class.

C#

```
public DuplicateWaitObjectException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Duplicate objects in argument." This message takes into account the current system culture.

The following table shows the initial property values for an instance of DuplicateWaitObjectException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DuplicateWaitObjectException(String)

Initializes a new instance of the DuplicateWaitObjectException class with the name of the parameter that causes this exception.

C#

```
public DuplicateWaitObjectException (string? parameterName);
```

Parameters

`parameterName` String

The name of the parameter that caused the exception.

Remarks

The content of `parameterName` is intended to be understood by humans.

The following table shows the initial property values for an instance of DuplicateWaitObjectException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.
ParamName	The parameter name string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DuplicateWaitObjectException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the DuplicateWaitObjectException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected DuplicateWaitObjectException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DuplicateWaitObjectException(String, Exception) *(Obsolete)*

Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public DuplicateWaitObjectException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed to this constructor.

The following table shows the initial property values for an instance of DuplicateWaitObjectException.

Expand table

Property	Value
InnerException	The exception specified by `innerException`.

Property	Value
Message	The error message string specified by `message`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DuplicateWaitObjectException(String, String)

Initializes a new instance of the DuplicateWaitObjectException class with a specified error message and the name of the parameter that causes this exception.

C#

```
public DuplicateWaitObjectException (string? parameterName, string?
message);
```

Parameters

`parameterName` String

The name of the parameter that caused the exception.

`message` String

The message that describes the error.

Remarks

The content of the `parameterName` and `message` parameters is intended to be understood by humans. The caller of this constructor is required to ensure that these strings have been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The

InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of DuplicateWaitObjectException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic)
Message	The error message string.
ParamName	The parameter name string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

EntryPointNotFoundException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt to load a class fails due to the absence of an entry method.

```
C#

public class EntryPointNotFoundException : TypeLoadException
```

Inheritance Object → Exception → SystemException → TypeLoadException → EntryPointNotFoundException

Remarks

An EntryPointNotFoundException exception is thrown when the common language runtime is unable to load an assembly because it cannot identify the assembly's entry point. This exception can be thrown under the following conditions:

- The common language runtime is unable to locate an application entry point (typically a `Main` method) in an executable assembly. The application entry point must be a global or `static` method that has either no parameters or a string array as its only parameter. The entry point can return `void`, or it can return an Int32 or UInt32 exit code. An application assembly cannot define more than one entry point.

- The call to a function in a Windows DLL cannot be resolved because the function cannot be found. In the following example, an EntryPointNotFoundException exception is thrown because User32.dll does not include a function named `GetMyNumber`.

  ```
  C#

  using System;
  using System.Runtime.InteropServices;
  ```

```
public class Example
{
   [DllImport("user32.dll")]
   public static extern int GetMyNumber();

   public static void Main()
   {
      try {
         int number = GetMyNumber();
      }
      catch (EntryPointNotFoundException e) {
         Console.WriteLine("{0}:\n   {1}", e.GetType().Name,
                           e.Message);
      }
   }
}
// The example displays the following output:
//    EntryPointNotFoundException:
//       Unable to find an entry point named 'GetMyNumber' in DLL
'User32.dll'.
```

- The call to a function in a Windows DLL cannot be resolved because the name used in the method call does not match a name found in the assembly. Frequently, this occurs because the DllImportAttribute.ExactSpelling field is either implicitly or explicitly set to `true`, the called method includes one or more string parameters and has both an ANSI and a Unicode version, and the name used in the method call does not correspond to the name of this ANSI or Unicode version. The following example provides an illustration by attempting to call the Windows `MessageBox` function in User32.dll. Because the first method definition specifies CharSet.Unicode for string marshaling, the common language looks for the wide-character version of the function, `MessageBoxW`, instead of the name used in the method call, `MessageBox`. The second method definition corrects this problem by calling the `MessageBoxW` instead of the `MessageBox` function.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
   [DllImport("user32.dll", CharSet = CharSet.Unicode, ExactSpelling =
true )]
   public static extern int MessageBox(IntPtr hwnd, String text, String
caption, uint type);

   [DllImport("user32.dll", CharSet = CharSet.Unicode, ExactSpelling =
true )]
```

```
   public static extern int MessageBoxW(IntPtr hwnd, String text,
String caption, uint type);

   public static void Main()
   {
      try {
         MessageBox(new IntPtr(0), "Calling the MessageBox Function",
"Example", 0);
      }
      catch (EntryPointNotFoundException e) {
         Console.WriteLine("{0}:\n   {1}", e.GetType().Name,
                           e.Message);
      }

      try {
         MessageBoxW(new IntPtr(0), "Calling the MessageBox Function",
"Example", 0);
      }
      catch (EntryPointNotFoundException e) {
         Console.WriteLine("{0}:\n   {1}", e.GetType().Name,
                           e.Message);
      }
   }
}
```

- You are trying to call a function in a dynamic link library by its simple name rather than its decorated name. Typically, the C++ compiler generates a decorated name for DLL functions. For example, the following C++ code defines a function named `Double` in a library named TestDll.dll.

```
C++

__declspec(dllexport) int Double(int number)
{
    return number * 2;
}
```

When the code in the following example tries to call the function, an EntryPointNotFoundException exception is thrown because the `Double` function cannot be found.

```
C#

using System;
using System.Runtime.InteropServices;

public class Example
{
   [DllImport("TestDll.dll")]
   public static extern int Double(int number);
```

```
    public static void Main()
    {
        Console.WriteLine(Double(10));
    }
}
// The example displays the following output:
//    Unhandled Exception: System.EntryPointNotFoundException: Unable
to find
//    an entry point named 'Double' in DLL '.\TestDll.dll'.
//       at Example.Double(Int32 number)
//       at Example.Main()
```

However, if the function is called by using its decorated name (in this case, `?Double@@YAHH@Z`), the function call succeeds, as the following example shows.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
    [DllImport("TestDll.dll", EntryPoint = "?Double@@YAHH@Z")]
    public static extern int Double(int number);

    public static void Main()
    {
        Console.WriteLine(Double(10));
    }
}
// The example displays the following output:
//    20
```

You can find the decorated names of functions exported by a DLL by using a utility such as Dumpbin.exe.

- You are attempting to call a method in a managed assembly as if it were an unmanaged dynamic link library. To see this in action, compile the following example to an assembly named StringUtilities.dll.

C#

```
using System;

public static class StringUtilities
{
    public static String SayGoodMorning(String name)
    {
        return String.Format("A top of the morning to you, {0}!", name);
```

```
    }
}
```

Then compile and execute the following example, which attempts to call the `StringUtilities.SayGoodMorning` method in the StringUtilities.dll dynamic link library as if it were unmanaged code. The result is an EntryPointNotFoundException exception.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
   [DllImport("StringUtilities.dll", CharSet = CharSet.Unicode )]
   public static extern String SayGoodMorning(String name);

   public static void Main()
   {
      Console.WriteLine(SayGoodMorning("Dakota"));
   }
}
// The example displays the following output:
//    Unhandled Exception: System.EntryPointNotFoundException: Unable
to find an entry point
//    named 'GoodMorning' in DLL 'StringUtilities.dll'.
//       at Example.GoodMorning(String& name)
//       at Example.Main()
```

To eliminate the exception, add a reference to the managed assembly and access the `StringUtilities.SayGoodMorning` method just as you would access any other method in managed code, as the following example does.

C#

```
using System;

public class Example
{
   public static void Main()
   {
       Console.WriteLine(StringUtilities.SayGoodMorning("Dakota"));
   }
}
// The example displays the following output:
//        A top of the morning to you, Dakota!
```

- You are trying to call a method in a COM DLL as if it were a Windows DLL. To access a COM DLL, select the **Add Reference** option in Visual Studio to add a reference to the project, and then select the type library from the **COM** tab.

For a list of initial property values for an instance of EntryPointNotFoundException, see the EntryPointNotFoundException constructors.

Constructors

Expand table

EntryPointNotFoundException()	Initializes a new instance of the EntryPointNotFoundException class.
EntryPointNotFound Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the EntryPointNotFoundException class with serialized data.
EntryPointNotFound Exception(String)	Initializes a new instance of the EntryPointNotFoundException class with a specified error message.
EntryPointNotFoundException(String, Exception)	Initializes a new instance of the EntryPointNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)

Message	Gets the error message for this exception. (Inherited from TypeLoadException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)
TypeName	Gets the fully qualified name of the type that causes the exception. (Inherited from TypeLoadException)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, method name, resource ID, and additional exception information. (Inherited from TypeLoadException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EntryPointNotFoundException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the EntryPointNotFoundException class.

Overloads

Expand table

EntryPointNotFoundException()	Initializes a new instance of the EntryPointNotFoundException class.
EntryPointNotFoundException(String)	Initializes a new instance of the EntryPointNotFoundException class with a specified error message.
EntryPointNotFoundException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the EntryPointNotFoundException class with serialized data.
EntryPointNotFoundException(String, Exception)	Initializes a new instance of the EntryPointNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

EntryPointNotFoundException()

Initializes a new instance of the EntryPointNotFoundException class.

C#

```
public EntryPointNotFoundException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Entry point was not found." This message takes into account the current system culture.

The following table shows the initial property values for an instance of EntryPointNotFoundException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

EntryPointNotFoundException(String)

Initializes a new instance of the EntryPointNotFoundException class with a specified error message.

C#

```
public EntryPointNotFoundException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of EntryPointNotFoundException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

EntryPointNotFoundException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the EntryPointNotFoundException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected EntryPointNotFoundException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

EntryPointNotFoundException(String, Exception)

Initializes a new instance of the EntryPointNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public EntryPointNotFoundException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of EntryPointNotFoundException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides the base class for enumerations.

C#

```
public abstract class Enum : ValueType, IComparable, IConvertible,
ISpanFormattable
```

Inheritance Object → ValueType → Enum

Derived Accessibility.AnnoScope
Microsoft.Aspnet.Snapin.MMC_CONTROL_TYPE
Microsoft.CSharp.ErrorLevel
Microsoft.CSharp.RuntimeBinder.CSharpArgumentInfoFlags
Microsoft.CSharp.RuntimeBinder.CSharpBinderFlags
More...

Implements IComparable , IConvertible , IFormattable , ISpanFormattable

Remarks

For more information about this API, see Supplemental API remarks for Enum.

Constructors

Expand table

Enum()	Initializes a new instance of the Enum class.

Methods

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Format(Type, Object, String)	Converts the specified value of a specified enumerated type to its equivalent string representation according to the specified format.
GetHashCode()	Returns the hash code for the value of this instance.
GetName(Type, Object)	Retrieves the name of the constant in the specified enumeration that has the specified value.
GetName<TEnum>(TEnum)	Retrieves the name of the constant in the specified enumeration type that has the specified value.
GetNames(Type)	Retrieves an array of the names of the constants in a specified enumeration.
GetNames<TEnum>()	Retrieves an array of the names of the constants in a specified enumeration type.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetTypeCode()	Returns the type code of the underlying type of this enumeration member.
GetUnderlyingType(Type)	Returns the underlying type of the specified enumeration.
GetValues(Type)	Retrieves an array of the values of the constants in a specified enumeration.
GetValues<TEnum>()	Retrieves an array of the values of the constants in a specified enumeration type.
GetValuesAsUnderlyingType(Type)	Retrieves an array of the values of the underlying type constants in a specified enumeration.
GetValuesAsUnderlyingType<TEnum>()	Retrieves an array of the values of the underlying type constants in a specified enumeration type.
HasFlag(Enum)	Determines whether one or more bit fields are set in the current instance.
IsDefined(Type, Object)	Returns a Boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.

IsDefined<TEnum>(TEnum)	Returns a boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Parse(Type, ReadOnlySpan<Char>)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
Parse(Type, ReadOnlySpan<Char>, Boolean)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse(Type, String)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
Parse(Type, String, Boolean)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse<TEnum>(ReadOnlySpan<Char>)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object.
Parse<TEnum>(ReadOnlySpan<Char>, Boolean)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse<TEnum>(String)	Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object.
Parse<TEnum>(String, Boolean)	Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
ToObject(Type, Byte)	Converts the specified 8-bit unsigned integer to an enumeration member.
ToObject(Type, Int16)	Converts the specified 16-bit signed integer to an enumeration member.
ToObject(Type, Int32)	Converts the specified 32-bit signed integer to an enumeration member.

ToObject(Type, Int64)	Converts the specified 64-bit signed integer to an enumeration member.
ToObject(Type, Object)	Converts the specified object with an integer value to an enumeration member.
ToObject(Type, SByte)	Converts the specified 8-bit signed integer value to an enumeration member.
ToObject(Type, UInt16)	Converts the specified 16-bit unsigned integer value to an enumeration member.
ToObject(Type, UInt32)	Converts the specified 32-bit unsigned integer value to an enumeration member.
ToObject(Type, UInt64)	Converts the specified 64-bit unsigned integer value to an enumeration member.
ToString()	Converts the value of this instance to its equivalent string representation.
ToString(IFormatProvider)	**Obsolete.** This method overload is obsolete; use ToString().
ToString(String)	Converts the value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormat Provider)	**Obsolete.** This method overload is obsolete; use ToString(String).
TryFormat<TEnum>(TEnum, Span<Char>, Int32, ReadOnly Span<Char>)	Tries to format the value of the enumerated type instance into the provided span of characters.
TryParse(Type, ReadOnly Span<Char>, Boolean, Object)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
TryParse(Type, ReadOnly Span<Char>, Object)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse(Type, String, Boolean, Object)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse(Type, String, Object)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

TryParse<TEnum>(ReadOnlySpan<Char>, Boolean, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-sensitive. The return value indicates whether the conversion succeeded.
TryParse<TEnum>(ReadOnlySpan<Char>, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse<TEnum>(String, Boolean, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-sensitive. The return value indicates whether the conversion succeeded.
TryParse<TEnum>(String, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. The return value indicates whether the conversion succeeded.

Explicit Interface Implementations

Expand table

IConvertible.ToBoolean(IFormatProvider)	Converts the current value to a Boolean value based on the underlying type.
IConvertible.ToByte(IFormatProvider)	Converts the current value to an 8-bit unsigned integer based on the underlying type.
IConvertible.ToChar(IFormatProvider)	Converts the current value to a Unicode character based on the underlying type.
IConvertible.ToDateTime(IFormatProvider)	Converts the current value to a DateTime based on the underlying type.
IConvertible.ToDecimal(IFormatProvider)	Converts the current value to a Decimal based on the underlying type.
IConvertible.ToDouble(IFormatProvider)	Converts the current value to a double-precision floating point number based on the underlying type.
IConvertible.ToInt16(IFormatProvider)	Converts the current value to a 16-bit signed integer based on the underlying type.

IConvertible.ToInt32(IFormatProvider)	Converts the current value to a 32-bit signed integer based on the underlying type.
IConvertible.ToInt64(IFormatProvider)	Converts the current value to a 64-bit signed integer based on the underlying type.
IConvertible.ToSByte(IFormatProvider)	Converts the current value to an 8-bit signed integer based on the underlying type.
IConvertible.ToSingle(IFormatProvider)	Converts the current value to a single-precision floating-point number based on the underlying type.
IConvertible.ToType(Type, IFormatProvider)	Converts the current value to a specified type based on the underlying type.
IConvertible.ToUInt16(IFormatProvider)	Converts the current value to a 16-bit unsigned integer based on the underlying type.
IConvertible.ToUInt32(IFormatProvider)	Converts the current value to a 32-bit unsigned integer based on the underlying type.
IConvertible.ToUInt64(IFormatProvider)	Converts the current value to a 64-bit unsigned integer based on the underlying type.
ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the enum into the provided span of characters.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

This type is thread safe.

See also

- ValueType
- FlagsAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Enum class.

C#

```
protected Enum ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object and returns an indication of their relative values.

```
C#
public int CompareTo (object? target);
```

Parameters

target Object

An object to compare, or `null`.

Returns

Int32

A signed number that indicates the relative values of this instance and `target`.

Expand table

Value	Meaning
Less than zero	The value of this instance is less than the value of `target`.
Zero	The value of this instance is equal to the value of `target`.
Greater than zero	The value of this instance is greater than the value of `target`, or `target` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`target` and this instance are not the same type.

InvalidOperationException

This instance is not type SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, or UInt64.

NullReferenceException

This instance is null.

Examples

The following example illustrates the use of `CompareTo` in the context of `Enum`.

C#

```
using System;

public class CompareToTest {
    enum VehicleDoors { Motorbike = 0, Sportscar = 2, Sedan = 4, Hatchback =
5 };

    public static void Main() {
        VehicleDoors myVeh = VehicleDoors.Sportscar;
        VehicleDoors yourVeh = VehicleDoors.Motorbike;
        VehicleDoors otherVeh = VehicleDoors.Sedan;

        Console.WriteLine("Does a {0} have more doors than a {1}?", myVeh,
yourVeh);
        Console.WriteLine( "{0}{1}", myVeh.CompareTo(yourVeh) > 0 ? "Yes" :
"No", Environment.NewLine );

        Console.WriteLine("Does a {0} have more doors than a {1}?", myVeh,
otherVeh);
        Console.WriteLine( "{0}", myVeh.CompareTo(otherVeh) > 0 ? "Yes" :
"No" );
    }
}
// The example displays the following output:
//       Does a Sportscar have more doors than a Motorbike?
//       Yes
//
//       Does a Sportscar have more doors than a Sedan?
//       No
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance, or `null`.

Returns

Boolean

`true` if `obj` is an enumeration value of the same type and with the same underlying value as this instance; otherwise, `false`.

Examples

The following example illustrates the use of the Equals method.

C#

```
using System;

public class EqualsTest {
    enum Colors { Red, Green, Blue, Yellow };
    enum Mammals { Cat, Dog, Horse, Dolphin };

    public static void Main() {
        Mammals myPet = Mammals.Cat;
        Colors myColor = Colors.Red;
        Mammals yourPet = Mammals.Dog;
        Colors yourColor = Colors.Red;
```

```
        Console.WriteLine("My favorite animal is a {0}", myPet);
        Console.WriteLine("Your favorite animal is a {0}", yourPet);
        Console.WriteLine("Do we like the same animal? {0}",
myPet.Equals(yourPet) ? "Yes" : "No");

        Console.WriteLine();
        Console.WriteLine("My favorite color is {0}", myColor);
        Console.WriteLine("Your favorite color is {0}", yourColor);
        Console.WriteLine("Do we like the same color? {0}",
myColor.Equals(yourColor) ? "Yes" : "No");

        Console.WriteLine();
        Console.WriteLine("The value of my color ({0}) is {1}", myColor,
Enum.Format(typeof(Colors), myColor, "d"));
        Console.WriteLine("The value of my pet (a {0}) is {1}", myPet,
Enum.Format(typeof(Mammals), myPet, "d"));
        Console.WriteLine("Even though they have the same value, are they
equal? {0}",
                    myColor.Equals(myPet) ? "Yes" : "No");
    }
}
// The example displays the following output:
//    My favorite animal is a Cat
//    Your favorite animal is a Dog
//    Do we like the same animal? No
//
//    My favorite color is Red
//    Your favorite color is Red
//    Do we like the same color? Yes
//
//    The value of my color (Red) is 0
//    The value of my pet (a Cat) is 0
//    Even though they have the same value, are they equal? No
```

The following example defines two enumeration types, `SledDog` and `WorkDog`. The `SledDog` enumeration has two members, `SledDog.AlaskanMalamute` and `SledDog.Malamute`, that have the same underlying value. The call to the Equals method indicates that these values are equal because their underlying values are the same. The `SledDog.Malamute` and `WorkDog.Newfoundland` members have the same underlying value, although they represent different enumeration types. A call to the Equals method indicates that these values are not equal.

C#

```
using System;

public enum SledDog { Unknown=0, AlaskanMalamute=1, Malamute=1,
                      Husky=2, SiberianHusky=2 };

public enum WorkDog { Unknown=0, Newfoundland=1, GreatPyrennes=2 };
```

```
public class Example
{
   public static void Main()
   {
      SledDog dog1 = SledDog.Malamute;
      SledDog dog2 = SledDog.AlaskanMalamute;
      WorkDog dog3 = WorkDog.Newfoundland;

      Console.WriteLine("{0:F} ({0:D}) = {1:F} ({1:D}): {2}",
                        dog1, dog2, dog1.Equals(dog2));
      Console.WriteLine("{0:F} ({0:D}) = {1:F} ({1:D}): {2}",
                        dog1, dog3, dog1.Equals(dog3));
   }
}
// The example displays the following output:
//       Malamute (1) = Malamute (1): True
//       Malamute (1) = Newfoundland (1): False
```

Remarks

The Enum.Equals(Object) method overrides ValueType.Equals(Object) to define how enumeration members are evaluated for equality.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- CompareTo(Object)

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Enum.Format(Type, Object, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified value of a specified enumerated type to its equivalent string representation according to the specified format.

C#

```
public static string Format (Type enumType, object value, string format);
```

Parameters

`enumType` Type

The enumeration type of the value to convert.

`value` Object

The value to convert.

`format` String

The output format to use.

Returns

String

A string representation of `value`.

Exceptions

ArgumentNullException

The `enumType`, `value`, or `format` parameter is `null`.

ArgumentException

The `enumType` parameter is not an Enum type.

-or-

The `value` is from an enumeration that differs in type from `enumType`.

-or-

The type of `value` is not an underlying type of `enumType`.

FormatException

The `format` parameter contains an invalid value.

InvalidOperationException

`format` equals "X", but the enumeration type is unknown.

-or-

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example illustrates the use of `Format` in the context of `Enum`.

C#

```
using System;

enum Colors { Red, Green, Blue, Yellow };

public class FormatTest {
    public static void Main() {
        Colors myColor = Colors.Blue;

        Console.WriteLine("My favorite color is {0}.", myColor);
        Console.WriteLine("The value of my favorite color is {0}.",
Enum.Format(typeof(Colors), myColor, "d"));
        Console.WriteLine("The hex value of my favorite color is {0}.",
Enum.Format(typeof(Colors), myColor, "x"));
    }
}
// The example displays the following output:
//    My favorite color is Blue.
//    The value of my favorite color is 2.
//    The hex value of my favorite color is 00000002.
```

Remarks

The following table shows the valid values for the `format` parameter.

Expand table

Format	Description
"G" or "g"	If `value` is equal to a named enumerated constant, the name of that constant is returned; otherwise, the decimal equivalent of `value` is returned. For example, suppose the only enumerated constant is named Red, and its value is 1. If `value` is specified as 1, this format returns "Red". However, if `value` is specified as 2, this format returns "2". -or- If the FlagsAttribute custom attribute is applied to the enumeration, `value` is treated as a bit field that contains one or more flags that consist of one or more bits. If `value` is equal to a combination of named enumerated constants, a delimiter-separated list of the names of those constants is returned. `value` is searched for flags, going from the flag with the largest value to the smallest value. For each flag that corresponds to a bit field in `value`, the name of the constant is concatenated to the delimiter-separated list. The value of that flag is then excluded from further consideration, and the search continues for the next flag. If `value` is not equal to a combination of named enumerated constants, the decimal equivalent of `value` is returned.
"X" or "x"	Represents `value` in hexadecimal format without a leading "0x".
"D" or "d"	Represents `value` in decimal form.
"F" or "f"	Behaves identically to "G" or "g", except that the FlagsAttribute is not required to be present on the Enum declaration.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToString(String)
- Formatting Types in .NET
- Enumeration Format Strings

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the value of this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Enum.GetName Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetName(Type, Object)	Retrieves the name of the constant in the specified enumeration that has the specified value.
GetName<TEnum>(TEnum)	Retrieves the name of the constant in the specified enumeration type that has the specified value.

GetName(Type, Object)

Retrieves the name of the constant in the specified enumeration that has the specified value.

C#

```
public static string? GetName (Type enumType, object value);
```

Parameters

`enumType` Type

An enumeration type.

`value` Object

The value of a particular enumerated constant in terms of its underlying type.

Returns

String

A string containing the name of the enumerated constant in `enumType` whose value is `value`; or `null` if no such constant is found.

Exceptions

ArgumentNullException

`enumType` or `value` is `null`.

ArgumentException

`enumType` is not an Enum.

-or-

`value` is neither of type `enumType` nor does it have the same underlying type as `enumType`.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example illustrates the use of `GetName`.

C#

```
using System;

public class GetNameTest {
    enum Colors { Red, Green, Blue, Yellow };
    enum Styles { Plaid, Striped, Tartan, Corduroy };

    public static void Main() {

        Console.WriteLine("The 4th value of the Colors Enum is {0}",
Enum.GetName(typeof(Colors), 3));
        Console.WriteLine("The 4th value of the Styles Enum is {0}",
Enum.GetName(typeof(Styles), 3));
    }
}
// The example displays the following output:
//       The 4th value of the Colors Enum is Yellow
//       The 4th value of the Styles Enum is Corduroy
```

Remarks

If multiple enumeration members have the same underlying value, the GetName method guarantees that it will return the name of one of those enumeration members. However, it does not guarantee that it will always return the name of the same enumeration member. As a result, when multiple enumeration members have the same value, your application code should never depend on the method returning a particular member's name.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetName<TEnum>(TEnum)

Retrieves the name of the constant in the specified enumeration type that has the specified value.

C#

```
public static string? GetName<TEnum> (TEnum value) where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the enumeration.

Parameters

`value` TEnum

The value of a particular enumerated constant in terms of its underlying type.

Returns

String

A string containing the name of the enumerated constant in `TEnum` whose value is `value`; or `null` if no such constant is found.

Exceptions

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.GetNames Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetNames(Type)	Retrieves an array of the names of the constants in a specified enumeration.
GetNames<TEnum>()	Retrieves an array of the names of the constants in a specified enumeration type.

GetNames(Type)

Retrieves an array of the names of the constants in a specified enumeration.

C#

```
public static string[] GetNames (Type enumType);
```

Parameters

enumType Type

An enumeration type.

Returns

String[]

A string array of the names of the constants in `enumType`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` parameter is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example illustrates the use of the GetNames method.

C#

```
using System;

public class GetNamesTest {
    enum Colors { Red, Green, Blue, Yellow };
    enum Styles { Plaid, Striped, Tartan, Corduroy };

    public static void Main() {

        Console.WriteLine("The members of the Colors enum are:");
        foreach(string s in Enum.GetNames(typeof(Colors)))
            Console.WriteLine(s);

        Console.WriteLine();

        Console.WriteLine("The members of the Styles enum are:");
        foreach(string s in Enum.GetNames(typeof(Styles)))
            Console.WriteLine(s);
    }
}
// The example displays the following output:
//       The members of the Colors enum are:
//       Red
//       Green
//       Blue
//       Yellow
//
//       The members of the Styles enum are:
//       Plaid
//       Striped
//       Tartan
//       Corduroy
```

Remarks

The elements of the return value array are sorted by the binary values of the enumerated constants (that is, by their unsigned magnitude). The following example provides displays information about the array returned by the GetNames method for an enumeration that includes a negative, zero, and a positive value.

C#

```
using System;

enum SignMagnitude { Negative = -1, Zero = 0, Positive = 1 };

public class Example
{
   public static void Main()
   {
      foreach (var name in Enum.GetNames(typeof(SignMagnitude))) {
         Console.WriteLine("{0,3:D}     0x{0:X}     {1}",
                           Enum.Parse(typeof(SignMagnitude), name),
                           name);
}   }
}
// The example displays the following output:
//          0     0x00000000     Zero
//          1     0x00000001     Positive
//         -1     0xFFFFFFFF     Negative
```

If there are enumerated constants with same value, the order of their corresponding names is unspecified.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetNames<TEnum>()

Retrieves an array of the names of the constants in a specified enumeration type.

```
C#

public static string[] GetNames<TEnum> () where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the enumeration.

Returns

String[]

A string array of the names of the constants in `TEnum`.

Exceptions

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the type code of the underlying type of this enumeration member.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The type code of the underlying type of this instance.

Implements

GetTypeCode()

Exceptions

InvalidOperationException

The enumeration type is unknown.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetUnderlyingType(Type)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.GetUnderlyingType(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the underlying type of the specified enumeration.

```
C#
public static Type GetUnderlyingType (Type enumType);
```

Parameters

enumType Type

The enumeration whose underlying type will be retrieved.

Returns

Type

The underlying type of `enumType`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

Examples

The following example calls the GetUnderlyingType method to display the underlying type of some enumeration members.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      Enum[] enumValues = { ConsoleColor.Red, DayOfWeek.Monday,
                            MidpointRounding.ToEven, PlatformID.Win32NT,
                            DateTimeKind.Utc, StringComparison.Ordinal };
      Console.WriteLine("{0,-10} {1, 18}   {2,15}\n",
                        "Member", "Enumeration", "Underlying Type");
      foreach (var enumValue in enumValues)
         DisplayEnumInfo(enumValue);
   }

   static void DisplayEnumInfo(Enum enumValue)
   {
      Type enumType = enumValue.GetType();
      Type underlyingType = Enum.GetUnderlyingType(enumType);
      Console.WriteLine("{0,-10} {1, 18}   {2,15}",
                        enumValue, enumType.Name, underlyingType.Name);
   }
}
// The example displays the following output:
//       Member            Enumeration   Underlying Type
//
//       Red              ConsoleColor             Int32
//       Monday              DayOfWeek             Int32
//       ToEven       MidpointRounding             Int32
//       Win32NT            PlatformID             Int32
//       Utc              DateTimeKind             Int32
//       Ordinal      StringComparison             Int32
```

Remarks

The Enum structure enables values to be represented as named constants. The data type of the enumeration's values is known as its underlying type. For example, the underlying type of the DayOfWeek enumeration, which consists of constants that represent each day of the week (DayOfWeek.Monday, DayOfWeek.Tuesday, and so on), is Int32.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetTypeCode()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.GetValues Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetValues(Type)	Retrieves an array of the values of the constants in a specified enumeration.
GetValues<TEnum>()	Retrieves an array of the values of the constants in a specified enumeration type.

GetValues(Type)

Retrieves an array of the values of the constants in a specified enumeration.

C#

```
public static Array GetValues (Type enumType);
```

Parameters

enumType Type

An enumeration type.

Returns

Array

An array that contains the values of the constants in `enumType`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

The method is invoked by reflection in a reflection-only context,

-or-

`enumType` is a type from an assembly loaded in a reflection-only context.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example illustrates the use of GetValues.

C#

```
using System;

public class GetValuesTest {
    enum Colors { Red, Green, Blue, Yellow };
    enum Styles { Plaid = 0, Striped = 23, Tartan = 65, Corduroy = 78 };

    public static void Main() {

        Console.WriteLine("The values of the Colors Enum are:");
        foreach(int i in Enum.GetValues(typeof(Colors)))
            Console.WriteLine(i);

        Console.WriteLine();

        Console.WriteLine("The values of the Styles Enum are:");
        foreach(int i in Enum.GetValues(typeof(Styles)))
            Console.WriteLine(i);
    }
}
// The example produces the following output:
//       The values of the Colors Enum are:
//       0
//       1
//       2
//       3
//
//       The values of the Styles Enum are:
```

```
//        0
//        23
//        65
//        78
```

Remarks

The elements of the array are sorted by the binary values of the enumeration constants (that is, by their unsigned magnitude). The following example displays information about the array returned by the GetValues method for an enumeration that includes a negative value, zero, and a positive value.

```C#
using System;

enum SignMagnitude { Negative = -1, Zero = 0, Positive = 1 };

public class Example
{
   public static void Main()
   {
      foreach (var value in Enum.GetValues(typeof(SignMagnitude))) {
         Console.WriteLine("{0,3}     0x{0:X8}     {1}",
                           (int) value, ((SignMagnitude) value));
}   }
}
// The example displays the following output:
//          0     0x00000000     Zero
//          1     0x00000001     Positive
//         -1     0xFFFFFFFF     Negative
```

The GetValues method returns an array that contains a value for each member of the `enumType` enumeration. If multiple members have the same value, the returned array includes duplicate values. In this case, calling the GetName method with each value in the returned array does not restore the unique names assigned to members that have duplicate values. To retrieve all the names of enumeration members successfully, call the GetNames method.

The GetValues method cannot be invoked by using reflection in a reflection-only context. Instead, you can retrieve the value of all enumeration members by using the Type.GetFields method to get an array of FieldInfo objects that represent enumeration members and then call the FieldInfo.GetRawConstantValue method on each element of the array. The following example illustrates this technique. It requires that you define the following enumeration in an assembly named Enumerations.dll:

C#

```
[Flags] enum Pets { None=0, Dog=1, Cat=2, Rodent=4, Bird=8,
                    Fish=16, Reptile=32, Other=64 };
```

The assembly is loaded in a reflection-only context, a Type object that represents the `Pets` enumeration is instantiated, an array of FieldInfo objects is retrieved, and the field values are displayed to the console.

C#

```
using System;
using System.Reflection;

public class Example
{
   public static void Main()
   {
      Assembly assem =
Assembly.ReflectionOnlyLoadFrom(@".\Enumerations.dll");
      Type typ = assem.GetType("Pets");
      FieldInfo[] fields = typ.GetFields();

      foreach (var field in fields) {
         if (field.Name.Equals("value__")) continue;

         Console.WriteLine("{0,-9} {1}", field.Name + ":",
                                         field.GetRawConstantValue());
      }
   }
}
// The example displays the following output:
//       None:     0
//       Dog:      1
//       Cat:      2
//       Rodent:   4
//       Bird:     8
//       Fish:     16
//       Reptile:  32
//       Other:    64
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetValues<TEnum>()

Retrieves an array of the values of the constants in a specified enumeration type.

C#

```
public static TEnum[] GetValues<TEnum> () where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the enumeration.

Returns

TEnum[]

An array that contains the values of the constants in `TEnum`.

Exceptions

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Enum.GetValuesAsUnderlyingType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetValuesAsUnderlyingType(Type)	Retrieves an array of the values of the underlying type constants in a specified enumeration.
GetValuesAsUnderlyingType<TEnum>()	Retrieves an array of the values of the underlying type constants in a specified enumeration type.

GetValuesAsUnderlyingType(Type)

Retrieves an array of the values of the underlying type constants in a specified enumeration.

C#

```
public static Array GetValuesAsUnderlyingType (Type enumType);
```

Parameters

enumType Type

An enumeration type.

Returns

Array

An array that contains the values of the underlying type constants in `enumType`.

Exceptions

ArgumentNullException

`enumType` is null.

ArgumentException

`enumType` is not an enumeration type.

Remarks

You can use this method to get enumeration values when it's hard to create an array of the enumeration type. For example, you might use this method for the MetadataLoadContext enumeration or on a platform where run-time code generation is not available.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

GetValuesAsUnderlyingType<TEnum>()

Retrieves an array of the values of the underlying type constants in a specified enumeration type.

C#

```
public static Array GetValuesAsUnderlyingType<TEnum> () where TEnum :
struct;
```

Type Parameters

`TEnum`

An enumeration type.

Returns

Array

An array that contains the values of the underlying type constants in `TEnum`.

Remarks

You can use this method to get enumeration values when it's hard to create an array of the enumeration type. For example, you might use this method for the MetadataLoadContext enumeration or on a platform where run-time code generation is not available.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.HasFlag(Enum) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one or more bit fields are set in the current instance.

```
C#
public bool HasFlag (Enum flag);
```

Parameters

`flag` Enum

An enumeration value.

Returns

Boolean

`true` if the bit field or bit fields that are set in `flag` are also set in the current instance; otherwise, `false`.

Exceptions

ArgumentException

`flag` is a different type than the current instance.

Examples

The following example defines an `DinnerItems` enumeration that reflects categories of items that a customer can order in a restaurant. The example tests whether the customer has ordered both an entrée and a beverage.

```
C#
```

```
using System;

[Flags] public enum DinnerItems {
   None = 0,
   Entree = 1,
   Appetizer = 2,
   Side = 4,
   Dessert = 8,
   Beverage = 16,
   BarBeverage = 32
}

public class Example
{
   public static void Main()
   {
      DinnerItems myOrder = DinnerItems.Appetizer | DinnerItems.Entree |
                            DinnerItems.Beverage | DinnerItems.Dessert;
      DinnerItems flagValue = DinnerItems.Entree | DinnerItems.Beverage;
      Console.WriteLine("{0} includes {1}: {2}",
                        myOrder, flagValue, myOrder.HasFlag(flagValue));
   }
}
// The example displays the following output:
//    Entree, Appetizer, Dessert, Beverage includes Entree, Beverage: True
```

Remarks

The HasFlag method returns the result of the following Boolean expression.

C#

```
(thisInstance & flag) == flag
```

If the underlying value of `flag` is zero, the method returns `true`. If this behavior is not desirable, you can use the Equals method to test for equality with zero and call HasFlag only if the underlying value of `flag` is non-zero, as the following example illustrates.

C#

```
using System;

[Flags] public enum Pets {
   None = 0,
   Dog = 1,
   Cat = 2,
   Bird = 4,
   Rabbit = 8,
```

```
    Other = 16
}

public class Example
{
    public static void Main()
    {
        Pets[] petsInFamilies = { Pets.None, Pets.Dog | Pets.Cat, Pets.Dog };
        int familiesWithoutPets = 0;
        int familiesWithDog = 0;

        foreach (var petsInFamily in petsInFamilies)
        {
            // Count families that have no pets.
            if (petsInFamily.Equals(Pets.None))
                familiesWithoutPets++;
            // Of families with pets, count families that have a dog.
            else if (petsInFamily.HasFlag(Pets.Dog))
                familiesWithDog++;
        }
        Console.WriteLine("{0} of {1} families in the sample have no pets.",
                          familiesWithoutPets, petsInFamilies.Length);
        Console.WriteLine("{0} of {1} families in the sample have a dog.",
                          familiesWithDog, petsInFamilies.Length);
    }
}
// The example displays the following output:
//       1 of 3 families in the sample have no pets.
//       2 of 3 families in the sample have a dog.
```

The HasFlag method is designed to be used with enumeration types that are marked with the FlagsAttribute attribute and can be used to determine whether multiple bit fields are set. For enumeration types that are not marked with the FlagsAttribute attribute, call either the Equals method or the CompareTo method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- FlagsAttribute
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IsDefined Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

IsDefined(Type, Object)	Returns a Boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.
IsDefined<TEnum>(TEnum)	Returns a boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.

IsDefined(Type, Object)

Returns a Boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.

C#

```
public static bool IsDefined (Type enumType, object value);
```

Parameters

enumType Type

An enumeration type.

value Object

The value or name of a constant in `enumType`.

Returns

Boolean

`true` if a constant in `enumType` has a value equal to `value`; otherwise, `false`.

Exceptions

ArgumentNullException

`enumType` or `value` is `null`.

ArgumentException

`enumType` is not an `Enum`.

-or-

The type of `value` is an enumeration, but it is not an enumeration of type `enumType`.

-or-

The type of `value` is not an underlying type of `enumType`.

InvalidOperationException

`value` is not type SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, or UInt64, or String.

Examples

The following example defines an enumeration named `PetType` that consists of individual bit fields. It then calls the IsDefined method with possible underlying enumeration values, string names, and composite values that result from setting multiple bit fields.

C#

```
using System;

[Flags] public enum PetType
{
   None = 0, Dog = 1, Cat = 2, Rodent = 4, Bird = 8, Reptile = 16, Other
= 32
};

public class Example
{
   public static void Main()
   {
      object value;

      // Call IsDefined with underlying integral value of member.
      value = 1;
```

```
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        // Call IsDefined with invalid underlying integral value.
        value = 64;
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        // Call IsDefined with string containing member name.
        value = "Rodent";
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        // Call IsDefined with a variable of type PetType.
        value = PetType.Dog;
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        value = PetType.Dog | PetType.Cat;
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        // Call IsDefined with uppercase member name.
        value = "None";
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        value = "NONE";
        Console.WriteLine("{0}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        // Call IsDefined with combined value
        value = PetType.Dog | PetType.Bird;
        Console.WriteLine("{0:D}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
        value = value.ToString();
        Console.WriteLine("{0:D}: {1}", value,
Enum.IsDefined(typeof(PetType), value));
    }
}
// The example displays the following output:
//       1: True
//       64: False
//       Rodent: True
//       Dog: True
//       Dog, Cat: False
//       None: True
//       NONE: False
//       9: False
//       Dog, Bird: False
```

Remarks

The `value` parameter can be any of the following:

- Any member of type `enumType`.
- A variable whose value is an enumeration member of type `enumType`.

- The string representation of the name of an enumeration member. The characters in the string must have the same case as the enumeration member name.
- A value of the underlying type of `enumType`.

If the constants in `enumType` define a set of bit fields and `value` contains the values, names, or underlying values of multiple bit fields, the IsDefined method returns `false`. In other words, for enumerations that define a set of bit fields, the method determines only whether a single bit field belongs to the enumeration. To determine whether multiple bit fields are set in an enumeration type that is tagged with the FlagsAttribute attribute, you can call the HasFlag method.

Notes to Callers

If `enumType` is an enumeration that is defined by using the FlagsAttribute attribute, the method returns `false` if multiple bit fields in `value` are set but `value` does not correspond to a composite enumeration value, or if `value` is a string concatenation of the names of multiple bit flags. In the following example, a `Pets` enumeration is defined with the FlagsAttribute attribute. The IsDefined(Type, Object) method returns `false` when you pass it an enumeration value that has two bit fields (`Pets.Dog` and `Pets.Cat`) set, and when you pass it the string representation of that enumeration value ("Dog, Cat").

C#

```
using System;

[Flags] public enum Pets {
      None = 0, Dog = 1, Cat = 2, Bird = 4,
      Rodent = 8, Other = 16 };

public class Example
{
   public static void Main()
   {
      Pets value = Pets.Dog | Pets.Cat;
      Console.WriteLine("{0:D} Exists: {1}",
                        value, Pets.IsDefined(typeof(Pets), value));
      string name = value.ToString();
      Console.WriteLine("{0} Exists: {1}",
                        name, Pets.IsDefined(typeof(Pets), name));
   }
}
// The example displays the following output:
```

```
//       3 Exists: False
//       Dog, Cat Exists: False
```

You can determine whether multiple bit fields are set by calling the HasFlag(Enum) method.

See also

- GetName(Type, Object)
- GetNames(Type)
- GetValues(Type)
- HasFlag(Enum)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsDefined<TEnum>(TEnum)

Returns a boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.

C#

```
public static bool IsDefined<TEnum> (TEnum value) where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the enumeration.

Parameters

`value` TEnum

The value or name of a constant in `TEnum`.

Returns

Boolean

`true` if a given integral value, or its name as a string, exists in a specified enumeration; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

Overloads

Expand table

Parse(Type, ReadOnlySpan<Char>)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
Parse(Type, String)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
Parse(Type, ReadOnlySpan<Char>, Boolean)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse(Type, String, Boolean)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse<TEnum>(String, Boolean)	Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse<TEnum>(ReadOnlySpan<Char>, Boolean)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
Parse<TEnum>(ReadOnlySpan<Char>)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by

	`TEnum` to an equivalent enumerated object.
Parse<TEnum>(String)	Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object.

Parse(Type, ReadOnlySpan<Char>)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

C#

```
public static object Parse (Type enumType, ReadOnlySpan<char> value);
```

Parameters

`enumType` Type

An enumeration type.

`value` ReadOnlySpan<Char>

A span containing the name or value to convert.

Returns

Object

An object of type `enumType` whose value is represented by `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

ArgumentException

`value` is either an empty string or only contains white space.

ArgumentException

`value` is a name, but not one of the named constants defined for the enumeration.

OverflowException

`value` is outside the range of the underlying type of `enumType`.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(Type, String)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

C#

```
public static object Parse (Type enumType, string value);
```

Parameters

`enumType` Type

An enumeration type.

`value` String

A string containing the name or value to convert.

Returns

Object

An object of type `enumType` whose value is represented by `value`.

Exceptions

ArgumentNullException

`enumType` or `value` is `null`.

ArgumentException

`enumType` is not an Enum.

-or-

`value` is either an empty string or only contains white space.

-or-

`value` is a name, but not one of the named constants defined for the enumeration.

OverflowException

`value` is outside the range of the underlying type of `enumType`.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example uses the Parse(Type, String) method to parse an array of strings that are created by calling the GetNames method. It also uses the Parse(Type, String) method to parse an enumeration value that consists of a bit field.

C#

```
using System;

public class ParseTest
{
    [Flags]
    enum Colors { Red = 1, Green = 2, Blue = 4, Yellow = 8 };

    public static void Main()
    {
        Console.WriteLine("The entries of the Colors enumeration are:");
        foreach (string colorName in Enum.GetNames(typeof(Colors)))
        {
            Console.WriteLine("{0} = {1:D}", colorName,
                                                Enum.Parse(typeof(Colors),
colorName));
        }
        Console.WriteLine();

        Colors orange = (Colors) Enum.Parse(typeof(Colors), "Red,
```

```
Yellow");
        Console.WriteLine("The orange value {0:D} has the combined
entries of {0}",
                          orange);
    }
}

/*
This code example produces the following results:

The entries of the Colors Enum are:
Red = 1
Green = 2
Blue = 4
Yellow = 8

The orange value 9 has the combined entries of Red, Yellow

*/
```

Remarks

The `value` parameter contains the string representation of an enumeration member's underlying value or named constant, or a list of named constants delimited by commas (,). One or more blank spaces can precede or follow each value, name, or comma in `value`. If `value` is a list, the return value is the value of the specified names combined with a bitwise `OR` operation.

If `value` is a name that does not correspond to a named constant of `enumType`, the method throws an ArgumentException. If `value` is the string representation of an integer that does not represent an underlying value of the `enumType` enumeration, the method returns an enumeration member whose underlying value is `value` converted to an integral type. If this behavior is undesirable, call the IsDefined method to ensure that a particular string representation of an integer is actually a member of `enumType`. The following example defines a `Colors` enumeration, calls the Parse(Type, String) method to convert strings to their corresponding enumeration values, and calls the IsDefined method to ensure that particular integral values are underlying values in the `Colors` enumeration.

C#

```
using System;

[Flags] enum Colors { None=0, Red = 1, Green = 2, Blue = 4 };

public class Example
```

```
{
   public static void Main()
   {
      string[] colorStrings = { "0", "2", "8", "blue", "Blue", "Yellow",
"Red, Green" };
      foreach (string colorString in colorStrings)
      {
         try {
            Colors colorValue = (Colors) Enum.Parse(typeof(Colors),
colorString);
            if (Enum.IsDefined(typeof(Colors), colorValue) |
colorValue.ToString().Contains(","))
               Console.WriteLine("Converted '{0}' to {1}.", colorString,
colorValue.ToString());
            else
               Console.WriteLine("{0} is not an underlying value of the
Colors enumeration.", colorString);
         }
         catch (ArgumentException) {
            Console.WriteLine("'{0}' is not a member of the Colors
enumeration.", colorString);
         }
      }
   }
}
// The example displays the following output:
//       Converted '0' to None.
//       Converted '2' to Green.
//       8 is not an underlying value of the Colors enumeration.
//       'blue' is not a member of the Colors enumeration.
//       Converted 'Blue' to Blue.
//       'Yellow' is not a member of the Colors enumeration.
//       Converted 'Red, Green' to Red, Green.
```

This operation is case-sensitive.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(Type, ReadOnlySpan<Char>, Boolean)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.

C#

```
public static object Parse (Type enumType, ReadOnlySpan<char> value, bool
ignoreCase);
```

Parameters

`enumType` Type

An enumeration type.

`value` ReadOnlySpan<Char>

A span containing the name or value to convert.

`ignoreCase` Boolean

`true` to ignore case; `false` to regard case.

Returns

Object

An object of type `enumType` whose value is represented by `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

ArgumentException

`value` is either an empty string or only contains white space.

ArgumentException

`value` is a name, but not one of the named constants defined for the enumeration.

OverflowException

`value` is outside the range of the underlying type of `enumType`

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(Type, String, Boolean)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.

C#

```
public static object Parse (Type enumType, string value, bool
ignoreCase);
```

Parameters

`enumType` Type

An enumeration type.

`value` String

A string containing the name or value to convert.

`ignoreCase` Boolean

`true` to ignore case; `false` to regard case.

Returns

Object

An object of type `enumType` whose value is represented by `value`.

Exceptions

ArgumentNullException

`enumType` or `value` is `null`.

ArgumentException

`enumType` is not an Enum.

-or-

`value` is either an empty string ("") or only contains white space.

-or-

`value` is a name, but not one of the named constants defined for the enumeration.

OverflowException

`value` is outside the range of the underlying type of `enumType`.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Examples

The following example uses the Parse(Type, String, Boolean) method to parse an array of strings that are created by calling the GetNames method. It also uses the Parse(Type, String) method to parse an enumeration value that consists of a bit field.

C#

```
using System;
```

```
[Flags] enum Colors { None=0, Red = 1, Green = 2, Blue = 4 };

public class Example
{
   public static void Main()
   {
      string[] colorStrings = { "0", "2", "8", "blue", "Blue", "Yellow",
"Red, Green" };
      foreach (string colorString in colorStrings)
      {
         try {
            Colors colorValue = (Colors) Enum.Parse(typeof(Colors),
colorString, true);
            if (Enum.IsDefined(typeof(Colors), colorValue) |
colorValue.ToString().Contains(","))
               Console.WriteLine("Converted '{0}' to {1}.", colorString,
colorValue.ToString());
            else
               Console.WriteLine("{0} is not an underlying value of the
Colors enumeration.", colorString);
         }
         catch (ArgumentException) {
            Console.WriteLine("{0} is not a member of the Colors
enumeration.", colorString);
         }
      }
   }
}
// The example displays the following output:
//       Converted '0' to None.
//       Converted '2' to Green.
//       8 is not an underlying value of the Colors enumeration.
//       Converted 'blue' to Blue.
//       Converted 'Blue' to Blue.
//       Yellow is not a member of the Colors enumeration.
//       Converted 'Red, Green' to Red, Green.
```

Remarks

The `value` parameter contains the string representation of an enumeration member's underlying value or named constant, or a list of named constants delimited by commas (,). One or more blank spaces can precede or follow each value, name, or comma in `value`. If `value` is a list, the return value is the value of the specified names combined with a bitwise `OR` operation.

If `value` is a name that does not correspond to a named constant of `enumType`, the method throws an ArgumentException. If `value` is the string representation of an integer that does not represent an underlying value of the `enumType` enumeration, the method returns an enumeration member whose underlying value is `value`

converted to an integral type. If this behavior is undesirable, call the IsDefined method to ensure that a particular string representation of an integer is actually a member of `enumType`. The following example defines a `Colors` enumeration, calls the Parse(Type, String, Boolean) method to convert strings to their corresponding enumeration values, and calls the IsDefined method to ensure that particular integral values are underlying values in the `Colors` enumeration.

The `ignoreCase` parameter specifies whether this operation is case-sensitive.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse<TEnum>(String, Boolean)

Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.

C#

```
public static TEnum Parse<TEnum> (string value, bool ignoreCase) where
TEnum : struct;
```

Type Parameters

`TEnum`

An enumeration type.

Parameters

`value` String

A string containing the name or value to convert.

`ignoreCase` Boolean

`true` to ignore case; `false` to regard case.

Returns

TEnum

An object of type `TEnum` whose value is represented by `value`.

Exceptions

ArgumentException

`TEnum` is not an Enum type.

ArgumentNullException

`value` is `null`.

ArgumentException

`value` does not contain enumeration information.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse<TEnum>(ReadOnlySpan<Char>, Boolean)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.

C#

```
public static TEnum Parse<TEnum> (ReadOnlySpan<char> value, bool
ignoreCase) where TEnum : struct;
```

Type Parameters

`TEnum`

An enumeration type.

Parameters

`value` ReadOnlySpan<Char>

A span containing the name or value to convert.

`ignoreCase` Boolean

`true` to ignore case; `false` to regard case.

Returns

TEnum

`TEnum` An object of type `TEnum` whose value is represented by `value`.

Exceptions

ArgumentException

`TEnum` is not an Enum type.

ArgumentException

`value` does not contain enumeration information.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse<TEnum>(ReadOnlySpan<Char>)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object.

C#

```
public static TEnum Parse<TEnum> (ReadOnlySpan<char> value) where TEnum :
struct;
```

Type Parameters

`TEnum`

An enumeration type.

Parameters

`value` ReadOnlySpan<Char>

A span containing the name or value to convert.

Returns

TEnum

`TEnum` An object of type `TEnum` whose value is represented by `value`.

Exceptions

ArgumentException

`TEnum` is not an Enum type.

ArgumentException

`value` does not contain enumeration information.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse<TEnum>(String)

Converts the string representation of the name or numeric value of one or more enumerated constants specified by `TEnum` to an equivalent enumerated object.

C#

```
public static TEnum Parse<TEnum> (string value) where TEnum : struct;
```

Type Parameters

`TEnum`

An enumeration type.

Parameters

`value` String

A string containing the name or value to convert.

Returns

TEnum

An object of type `TEnum` whose value is represented by `value`.

Exceptions

ArgumentException

`TEnum` is not an Enum type.

ArgumentNullException

`value` is `null`.

ArgumentException

`value` does not contain enumeration information.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.ToObject Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified integer value to an enumeration member.

Overloads

Expand table

ToObject(Type, Int16)	Converts the specified 16-bit signed integer to an enumeration member.
ToObject(Type, Int32)	Converts the specified 32-bit signed integer to an enumeration member.
ToObject(Type, Int64)	Converts the specified 64-bit signed integer to an enumeration member.
ToObject(Type, Object)	Converts the specified object with an integer value to an enumeration member.
ToObject(Type, SByte)	Converts the specified 8-bit signed integer value to an enumeration member.
ToObject(Type, UInt16)	Converts the specified 16-bit unsigned integer value to an enumeration member.
ToObject(Type, UInt32)	Converts the specified 32-bit unsigned integer value to an enumeration member.
ToObject(Type, UInt64)	Converts the specified 64-bit unsigned integer value to an enumeration member.
ToObject(Type, Byte)	Converts the specified 8-bit unsigned integer to an enumeration member.

ToObject(Type, Int16)

Converts the specified 16-bit signed integer to an enumeration member.

```
C#
```

```
public static object ToObject (Type enumType, short value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` Int16

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, Int16) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, Int32)

Converts the specified 32-bit signed integer to an enumeration member.

```
C#

public static object ToObject (Type enumType, int value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` Int32

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, Int32) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, Int64)

Converts the specified 64-bit signed integer to an enumeration member.

C#

```
public static object ToObject (Type enumType, long value);
```

Parameters

enumType Type

The enumeration type to return.

`value` Int64

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, Int64) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, Object)

Converts the specified object with an integer value to an enumeration member.

```
C#
public static object ToObject (Type enumType, object value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` Object

The value convert to an enumeration member.

Returns

Object

An enumeration object whose value is `value`.

Exceptions

ArgumentNullException

`enumType` or `value` is `null`.

ArgumentException

`enumType` is not an Enum.

-or-

`value` is not type SByte, Int16, Int32, Int64, Byte, UInt16, UInt32, or UInt64.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, Object) method converts the integral value `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToObject(Type, SByte)

> Important
>
> This API is not CLS-compliant.

Converts the specified 8-bit signed integer value to an enumeration member.

```
C#

[System.CLSCompliant(false)]
public static object ToObject (Type enumType, sbyte value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` SByte

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, SByte) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, UInt16)

> Important
>
> This API is not CLS-compliant.

Converts the specified 16-bit unsigned integer value to an enumeration member.

C#

```
[System.CLSCompliant(false)]
public static object ToObject (Type enumType, ushort value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` UInt16

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, UInt16) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, UInt32)

> Important
>
> This API is not CLS-compliant.

Converts the specified 32-bit unsigned integer value to an enumeration member.

C#

```
[System.CLSCompliant(false)]
public static object ToObject (Type enumType, uint value);
```

Parameters

enumType Type

The enumeration type to return.

value UInt32

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, UInt32) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, UInt64)

> Important
>
> This API is not CLS-compliant.

Converts the specified 64-bit unsigned integer value to an enumeration member.

C#

```
[System.CLSCompliant(false)]
public static object ToObject (Type enumType, ulong value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` UInt64

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, UInt64) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToObject(Type, Byte)

Converts the specified 8-bit unsigned integer to an enumeration member.

```
C#

public static object ToObject (Type enumType, byte value);
```

Parameters

`enumType` Type

The enumeration type to return.

`value` Byte

The value to convert to an enumeration member.

Returns

Object

An instance of the enumeration set to `value`.

Exceptions

ArgumentNullException

`enumType` is `null`.

ArgumentException

`enumType` is not an Enum.

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Remarks

The ToObject(Type, Byte) method converts `value` to an enumeration member whose underlying value is `value`. Note that the conversion succeeds even if value is outside the bounds of `enumType` members. To ensure that `value` is a valid underlying value of the `enumType` enumeration, pass it to the IsDefined method.

This conversion method returns a value of type Object. You can then cast it or convert it to an object of type `enumType`.

See also

- IsDefined(Type, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to its equivalent string representation.

Overloads

Expand table

ToString(String, IFormatProvider)	**Obsolete.** This method overload is obsolete; use ToString(String).
ToString()	Converts the value of this instance to its equivalent string representation.
ToString(String)	Converts the value of this instance to its equivalent string representation using the specified format.
ToString(IFormatProvider)	**Obsolete.** This method overload is obsolete; use ToString().

ToString(String, IFormatProvider)

Caution

The provider argument is not used. Use ToString(String) instead.

This method overload is obsolete; use ToString(String).

C#

```
[System.Obsolete("The provider argument is not used. Use ToString(String) instead.")]
```

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A format specification.

`provider` IFormatProvider

(Obsolete.)

Returns

String

The string representation of the value of this instance as specified by `format`.

Implements

ToString(String, IFormatProvider)

Attributes ObsoleteAttribute

Exceptions

FormatException

`format` does not contain a valid format specification.

InvalidOperationException

`format` equals "X", but the enumeration type is unknown.

Remarks

The `format` parameter can be one of the following format strings: "G" or "g", "D" or "d", "X" or "x", and "F" or "f" (the format string is not case-sensitive). If `format` is `null` or an empty string (""), the general format specifier ("G") is used. For more information about the enumeration format strings and formatting enumeration values, see Enumeration Format Strings. For more information about formatting in general, see Formatting Types.

Specify only `format`; the `provider` parameter is obsolete.

See also

- Format(Type, Object, String)
- Formatting Types in .NET
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

ToString()

Converts the value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Examples

The following example demonstrates converting an enumerated value to a string.

C#

```
using System;

public class EnumSample {
    enum Colors {Red = 1, Blue = 2};
```

```
    public static void Main() {
        Enum myColors = Colors.Red;
        Console.WriteLine("The value of this instance is '{0}'",
           myColors.ToString());
    }
}
/*
Output.
The value of this instance is 'Red'.
*/
```

Remarks

The return value is formatted with the general format specifier ("G"). That is, if the FlagsAttribute is not applied to this enumerated type and there is a named constant equal to the value of this instance, then the return value is a string containing the name of the constant. If the FlagsAttribute is applied and there is a combination of one or more named constants equal to the value of this instance, then the return value is a string containing a delimiter-separated list of the names of the constants. Otherwise, the return value is the string representation of the numeric value of this instance. For more information about formatting enumeration values, see Enumeration Format Strings. For more information about formatting in general, see Formatting Types.

Notes to Callers

If multiple enumeration members have the same underlying value and you attempt to retrieve the string representation of an enumeration member's name based on its underlying value, your code should not make any assumptions about which name the method will return. For example, the following enumeration defines two members, `Shade.Gray` and `Shade.Grey`, that have the same underlying value.

C#

```
enum Shade
{
    White = 0, Gray = 1, Grey = 1, Black = 2
}
```

The following method call attempts to retrieve the name of a member of the `Shade` enumeration whose underlying value is 1. The method can return either "Gray" or "Grey", and your code should not make any assumptions about which string will be returned.

```
C#
string shadeName = ((Shade) 1).ToString();
```

See also

- Format(Type, Object, String)
- Parse(Type, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the value of this instance to its equivalent string representation using the specified format.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` contains an invalid specification.

InvalidOperationException

`format` equals "X", but the enumeration type is unknown.

Examples

The following example demonstrates how to convert an enumerated value to a string.

C#

```
// Sample for Enum.ToString(String)
using System;

class Sample
{
    enum Colors {Red, Green, Blue, Yellow = 12};

    public static void Main()
    {
    Colors myColor = Colors.Yellow;

    Console.WriteLine("Colors.Red = {0}", Colors.Red.ToString("d"));
    Console.WriteLine("Colors.Green = {0}", Colors.Green.ToString("d"));
    Console.WriteLine("Colors.Blue = {0}", Colors.Blue.ToString("d"));
    Console.WriteLine("Colors.Yellow = {0}",
Colors.Yellow.ToString("d"));

    Console.WriteLine("{0}myColor = Colors.Yellow{0}",
Environment.NewLine);

    Console.WriteLine("myColor.ToString(\"g\") = {0}",
myColor.ToString("g"));
    Console.WriteLine("myColor.ToString(\"G\") = {0}",
myColor.ToString("G"));

    Console.WriteLine("myColor.ToString(\"x\") = {0}",
myColor.ToString("x"));
    Console.WriteLine("myColor.ToString(\"X\") = {0}",
myColor.ToString("X"));

    Console.WriteLine("myColor.ToString(\"d\") = {0}",
myColor.ToString("d"));
```

```
    Console.WriteLine("myColor.ToString(\"D\") = {0}",
myColor.ToString("D"));

    Console.WriteLine("myColor.ToString(\"f\") = {0}",
myColor.ToString("f"));
    Console.WriteLine("myColor.ToString(\"F\") = {0}",
myColor.ToString("F"));
    }
}
/*
This example produces the following results:
Colors.Red = 0
Colors.Green = 1
Colors.Blue = 2
Colors.Yellow = 12

myColor = Colors.Yellow

myColor.ToString("g") = Yellow
myColor.ToString("G") = Yellow
myColor.ToString("x") = 0000000C
myColor.ToString("X") = 0000000C
myColor.ToString("d") = 12
myColor.ToString("D") = 12
myColor.ToString("f") = Yellow
myColor.ToString("F") = Yellow
*/
```

Remarks

The `format` parameter can be one of the following format strings: "G" or "g", "D" or "d", "X" or "x", and "F" or "f" (the format string is not case-sensitive). If `format` is `null` or an empty string (""), the general format specifier ("G") is used. For more information about the enumeration format strings and formatting enumeration values, see Enumeration Format Strings. For more information about formatting in general, see Formatting Types.

Notes to Callers

If multiple enumeration members have the same underlying value and you attempt to retrieve the string representation of an enumeration member's name based on its underlying value, your code should not make any assumptions about which name the method will return. For example, the following enumeration defines two members, `Shade.Gray` and `Shade.Grey`, that have the same underlying value.

```
C#
```

```
enum Shade
{
    White = 0, Gray = 1, Grey = 1, Black = 2
}
```

The following method call attempts to retrieve the name of a member of the `Shade` enumeration whose underlying value is 1. The method can return either "Gray" or "Grey", and your code should not make any assumptions about which string will be returned.

C#

```
string shadeName = ((Shade) 1).ToString("F");
```

See also

- Format(Type, Object, String)
- Parse(Type, String)
- ToString(String, IFormatProvider)
- Formatting Types in .NET
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

⊗ **Caution**

The provider argument is not used. Use ToString() instead.

This method overload is obsolete; use ToString().

C#

```
[System.Obsolete("The provider argument is not used. Use ToString()
instead.")]
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

(obsolete)

Returns

String

The string representation of the value of this instance.

Implements

ToString(IFormatProvider)

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Enum.TryFormat<TEnum> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the enumerated type instance into the provided span of characters.

C#

```
public static bool TryFormat<TEnum> (TEnum value, Span<char> destination,
out int charsWritten, ReadOnlySpan<char> format = default) where TEnum :
struct;
```

Type Parameters

TEnum

Parameters

value TEnum

destination Span<Char>

The span into which to write the instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the character that represents the standard format string that defines the acceptable format of destination. This value can be empty, or "g", "d", "f", or "x".

Returns

Boolean

`true` if the formatting was successful; otherwise, `false` if the destination span wasn't large enough to contain the formatted value.

Exceptions

FormatException

The format parameter contains an invalid value.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. The return value indicates whether the conversion succeeded.

Overloads

Expand table

TryParse(Type, ReadOnlySpan<Char>, Object)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse(Type, String, Object)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse(Type, ReadOnlySpan<Char>, Boolean, Object)	Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.
TryParse(Type, String, Boolean, Object)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse<TEnum>(ReadOnlySpan<Char>, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
TryParse<TEnum>(String, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. The return value indicates whether the conversion succeeded.
TryParse<TEnum>(String, Boolean, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation

	is case-sensitive. The return value indicates whether the conversion succeeded.
TryParse<TEnum> (ReadOnlySpan<Char>, Boolean, TEnum)	Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-sensitive. The return value indicates whether the conversion succeeded.

TryParse(Type, ReadOnlySpan<Char>, Object)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

```
C#

public static bool TryParse (Type enumType, ReadOnlySpan<char> value, out
object? result);
```

Parameters

`enumType` Type

The enum type to use for parsing.

`value` ReadOnlySpan<Char>

The span representation of the name or numeric value of one or more enumerated constants.

`result` Object

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(Type, String, Object)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

C#

```
public static bool TryParse (Type enumType, string? value, out object?
result);
```

Parameters

`enumType` Type

The enum type to use for parsing.

`value` String

The string representation of the name or numeric value of one or more enumerated constants.

`result` Object

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(Type, ReadOnlySpan<Char>, Boolean, Object)

Converts the span of characters representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-insensitive.

C#

```
public static bool TryParse (Type enumType, ReadOnlySpan<char> value,
bool ignoreCase, out object? result);
```

Parameters

`enumType` Type

The enum type to use for parsing.

`value` ReadOnlySpan<Char>

The span representation of the name or numeric value of one or more enumerated constants.

`ignoreCase` Boolean

`true` to read `enumType` in case insensitive mode; `false` to read `enumType` in case sensitive mode.

`result` Object

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(Type, String, Boolean, Object)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

C#

```
public static bool TryParse (Type enumType, string? value, bool
ignoreCase, out object? result);
```

Parameters

`enumType` Type

The enum type to use for parsing.

`value` String

The string representation of the name or numeric value of one or more enumerated constants.

`ignoreCase` Boolean

`true` to read `value` in case insensitive mode; `false` to read `value` in case sensitive mode.

`result` Object

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

InvalidOperationException

.NET 8 and later versions: `enumType` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse<TEnum>(ReadOnlySpan<Char>, TEnum)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

C#

```
public static bool TryParse<TEnum> (ReadOnlySpan<char> value, out TEnum
result) where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the `result` object.

Parameters

`value` ReadOnlySpan<Char>

The span representation of the name or numeric value of one or more enumerated constants.

`result` TEnum

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

ArgumentException

`TEnum` is not an enumeration type.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse<TEnum>(String, TEnum)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. The return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse<TEnum> (string? value, out TEnum result)
where TEnum : struct;
```

Type Parameters

`TEnum`

The enumeration type to which to convert `value`.

Parameters

`value` String

The case-sensitive string representation of the enumeration name or underlying value to convert.

`result` TEnum

When this method returns, contains an object of type `TEnum` whose value is represented by `value` if the parse operation succeeds. If the parse operation fails, contains the default value of the underlying type of `TEnum`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `value` parameter was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`TEnum` is not an enumeration type.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Examples

The following example defines a `Colors` enumeration, calls the TryParse<TEnum> (String, TEnum) method to convert strings to their corresponding enumeration values, and calls the IsDefined method to ensure that particular integral values are underlying values in the `Colors` enumeration.

C#

```
using System;

[Flags] enum Colors { None=0, Red = 1, Green = 2, Blue = 4 };
```

```
public class Example
{
   public static void Main()
   {
      string[] colorStrings = { "0", "2", "8", "blue", "Blue", "Yellow",
"Red, Green" };
      foreach (string colorString in colorStrings)
      {
         Colors colorValue;
         if (Enum.TryParse(colorString, out colorValue))
            if (Enum.IsDefined(typeof(Colors), colorValue) |
colorValue.ToString().Contains(","))
               Console.WriteLine("Converted '{0}' to {1}.", colorString,
colorValue.ToString());
            else
               Console.WriteLine("{0} is not an underlying value of the
Colors enumeration.", colorString);
         else
            Console.WriteLine("{0} is not a member of the Colors
enumeration.", colorString);
      }
   }
}
// The example displays the following output:
//       Converted '0' to None.
//       Converted '2' to Green.
//       8 is not an underlying value of the Colors enumeration.
//       blue is not a member of the Colors enumeration.
//       Converted 'Blue' to Blue.
//       Yellow is not a member of the Colors enumeration.
//       Converted 'Red, Green' to Red, Green.
```

Remarks

TryParse<TEnum>(String, TEnum) is identical to the Parse(Type, String) method, except that instead of throwing an exception, it returns `false` if the conversion fails. It eliminates the need for exception handling when parsing the string representation of an enumeration value.

The `value` parameter contains the string representation of an enumeration member's underlying value or named constant, or a list of named constants or underlying values delimited by commas (,). If `value` includes multiple named constants or values, one or more blank spaces can precede or follow each value, name, or comma in `value`. If `value` is a list, `result` reflects the value of the specified names or underlying values combined with a bitwise `OR` operation. If `value` is the string representation of the name of an enumeration value, the comparison of `value` with enumeration names is case-sensitive.

If `value` is a name that does not correspond to a named constant of `TEnum`, the method returns `false`. If `value` is the string representation of an integer that does not represent an underlying value of the `TEnum` enumeration, the method returns an enumeration member whose underlying value is `value` converted to an integral type. If this behavior is undesirable, call the IsDefined method to ensure that a particular string representation of an integer is actually a member of `TEnum`.

If the parse operation fails, the `result` parameter contains the default value of 0, which may not be a member of the underlying `TEnum` enumeration. If no values are assigned to the named constants in `TEnum`, the default is equal to the first member of the `TEnum`. Otherwise, the default is equal to the member in the enumeration with an assigned value of 0.

See also

- Parse(Type, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse<TEnum>(String, Boolean, TEnum)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-sensitive. The return value indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse<TEnum> (string? value, bool ignoreCase, out
TEnum result) where TEnum : struct;
```

Type Parameters

`TEnum`

The enumeration type to which to convert `value`.

Parameters

`value` String

The string representation of the enumeration name or underlying value to convert.

`ignoreCase` Boolean

`true` to ignore case; `false` to consider case.

`result` TEnum

When this method returns, contains an object of type `TEnum` whose value is represented by `value` if the parse operation succeeds. If the parse operation fails, contains the default value of the underlying type of `TEnum`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the `value` parameter was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`TEnum` is not an enumeration type.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Examples

The following example defines a `Colors` enumeration, calls the TryParse<TEnum> (String, Boolean, TEnum) method to convert strings to their corresponding

enumeration values, and calls the IsDefined method to ensure that particular integral values are underlying values in the `Colors` enumeration. The TryParse<TEnum> (String, Boolean, TEnum) method uses case-insensitive comparison when trying to convert the string representations of named constants to their equivalent enumeration values.

C#

```
using System;

[Flags] enum Colors { None=0, Red = 1, Green = 2, Blue = 4 };

public class Example
{
   public static void Main()
   {
      string[] colorStrings = { "0", "2", "8", "blue", "Blue", "Yellow",
"Red, Green" };
      foreach (string colorString in colorStrings)
      {
         Colors colorValue;
         if (Enum.TryParse(colorString, true, out colorValue))
            if (Enum.IsDefined(typeof(Colors), colorValue) |
colorValue.ToString().Contains(","))
               Console.WriteLine("Converted '{0}' to {1}.", colorString,
colorValue.ToString());
            else
               Console.WriteLine("{0} is not an underlying value of the
Colors enumeration.", colorString);
         else
            Console.WriteLine("{0} is not a member of the Colors
enumeration.", colorString);
      }
   }
}
// The example displays the following output:
//       Converted '0' to None.
//       Converted '2' to Green.
//       8 is not an underlying value of the Colors enumeration.
//       Converted 'blue' to Blue.
//       Converted 'Blue' to Blue.
//       Yellow is not a member of the Colors enumeration.
//       Converted 'Red, Green' to Red, Green.
```

Remarks

TryParse<TEnum>(String, Boolean, TEnum) is identical to the Parse(Type, String, Boolean) method, except that instead of throwing an exception, it returns `false` if

the conversion fails. It eliminates the need for exception handling when parsing the string representation of an enumeration value.

The `value` parameter contains the string representation of an enumeration member's underlying value or named constant, or a list of named constants or underlying values delimited by commas (,). If `value` includes multiple named constants or values, one or more blank spaces can precede or follow each value, name, or comma in `value`. If `value` is a list, `result` reflects the value of the specified names or underlying values combined with a bitwise `OR` operation. If `value` is the string representation of the name of an enumeration value, the comparison of `value` with enumeration names depends on the `ignoreCase` parameter. If `true`, the comparison is case-insensitive; if `false`, it is case-sensitive.

If `value` is a name that does not correspond to a named constant of `TEnum`, the method returns `false`. If `value` is the string representation of an integer that does not represent an underlying value of the `TEnum` enumeration, the method returns an enumeration member whose underlying value is `value` converted to an integral type. If this behavior is undesirable, call the IsDefined method to ensure that a particular string representation of an integer is actually a member of `TEnum`.

If the parse operation fails, the `result` parameter contains the default value of 0, which may not be a member of the underlying `TEnum` enumeration. If no values are assigned to the named constants in `TEnum`, the default is equal to the first member of the `TEnum`. Otherwise, the default is equal to the member in the enumeration with an assigned value of 0.

See also

- Parse(Type, String, Boolean)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse<TEnum>(ReadOnlySpan<Char>, Boolean, TEnum)

Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. A parameter specifies whether the operation is case-sensitive. The return value indicates whether the conversion succeeded.

```
C#

public static bool TryParse<TEnum> (ReadOnlySpan<char> value, bool
ignoreCase, out TEnum result) where TEnum : struct;
```

Type Parameters

`TEnum`

The type of the `result` object.

Parameters

`value` ReadOnlySpan<Char>

The span representation of the name or numeric value of one or more enumerated constants.

`ignoreCase` Boolean

`true` to ignore case; `false` to consider case.

`result` TEnum

When this method returns `true`, contains an enumeration constant that represents the parsed value.

Returns

Boolean

`true` if the conversion succeeded; `false` otherwise.

Exceptions

ArgumentException

`TEnum` is not an enumeration type.

InvalidOperationException

.NET 8 and later versions: `TEnum` is a Boolean-backed enumeration type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a Boolean value based on the underlying type.

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Boolean

This member always throws an exception.

Implements

ToBoolean(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to an 8-bit unsigned integer based on the underlying type.

C#

```
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Byte

The converted value.

Implements

ToByte(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a Unicode character based on the underlying type.

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Char

This member always throws an exception.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a DateTime based on the underlying type.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

DateTime

This member always throws an exception.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a Decimal based on the underlying type.

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Decimal

This member always throws an exception.

Implements

ToDecimal(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a double-precision floating point number based on the underlying type.

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Double

This member always throws an exception.

Implements

ToDouble(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a 16-bit signed integer based on the underlying type.

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int16

The converted value.

Implements

ToInt16(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a 32-bit signed integer based on the underlying type.

C#

```
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int32

The converted value.

Implements

ToInt32(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a 64-bit signed integer based on the underlying type.

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Int64

The converted value.

Implements

ToInt64(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the current value to an 8-bit signed integer based on the underlying type.

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

SByte

The converted value.

Implements

ToSByte(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a single-precision floating-point number based on the underlying type.

```
C#

float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Single

This member always throws an exception.

Implements

ToSingle(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Remarks

Currently not implemented. Always throws an exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current value to a specified type based on the underlying type.

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to convert to.

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

Object

The converted value.

Implements

ToType(Type, IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the current value to a 16-bit unsigned integer based on the underlying type.

```
C#
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt16

The converted value.

Implements

ToUInt16(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the current value to a 32-bit unsigned integer based on the underlying type.

```
C#
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt32

The converted value.

Implements

ToUInt32(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the current value to a 64-bit unsigned integer based on the underlying type.

```
C#
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

UInt64

The converted value.

Implements

ToUInt64(IFormatProvider)

Remarks

Uses Convert to perform the conversion.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enum.ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the enum into the provided span of characters.

C#

```
bool ISpanFormattable.TryFormat (Span<char> destination, out int
charsWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in destination.

format ReadOnlySpan<Char>

The format specifier.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for destination. This parameter is ignored.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides information about, and means to manipulate, the current environment and platform. This class cannot be inherited.

C#

```
public static class Environment
```

Inheritance Object → Environment

Examples

The following example displays a list of information about the current environment.

C#

```
// Sample for Environment class summary
using System;
using System.Collections;

class Sample
{
    public static void Main()
    {
        string str;
        string nl = Environment.NewLine;
        //
        Console.WriteLine();
        Console.WriteLine("-- Environment members --");

        //  Invoke this sample with an arbitrary set of command line
arguments.
        Console.WriteLine("CommandLine: {0}", Environment.CommandLine);

        string[] arguments = Environment.GetCommandLineArgs();
        Console.WriteLine("GetCommandLineArgs: {0}", String.Join(", ",
arguments));
```

```
        //  <-- Keep this information secure! -->
        Console.WriteLine("CurrentDirectory: {0}",
Environment.CurrentDirectory);

        Console.WriteLine("ExitCode: {0}", Environment.ExitCode);

        Console.WriteLine("HasShutdownStarted: {0}",
Environment.HasShutdownStarted);

        //  <-- Keep this information secure! -->
        Console.WriteLine("MachineName: {0}", Environment.MachineName);

        Console.WriteLine("NewLine: {0}  first line{0}  second line{0}
third line",
                              Environment.NewLine);

        Console.WriteLine("OSVersion: {0}",
Environment.OSVersion.ToString());

        Console.WriteLine("StackTrace: '{0}'", Environment.StackTrace);

        //  <-- Keep this information secure! -->
        Console.WriteLine("SystemDirectory: {0}",
Environment.SystemDirectory);

        Console.WriteLine("TickCount: {0}", Environment.TickCount);

        //  <-- Keep this information secure! -->
        Console.WriteLine("UserDomainName: {0}",
Environment.UserDomainName);

        Console.WriteLine("UserInteractive: {0}",
Environment.UserInteractive);

        //  <-- Keep this information secure! -->
        Console.WriteLine("UserName: {0}", Environment.UserName);

        Console.WriteLine("Version: {0}", Environment.Version.ToString());

        Console.WriteLine("WorkingSet: {0}", Environment.WorkingSet);

        //  No example for Exit(exitCode) because doing so would terminate
this example.

        //  <-- Keep this information secure! -->
        string query = "My system drive is %SystemDrive% and my system root
is %SystemRoot%";
        str = Environment.ExpandEnvironmentVariables(query);
        Console.WriteLine("ExpandEnvironmentVariables: {0}  {1}", nl, str);

        Console.WriteLine("GetEnvironmentVariable: {0}  My temporary
directory is {1}.", nl,
                               Environment.GetEnvironmentVariable("TEMP"));

        Console.WriteLine("GetEnvironmentVariables: ");
```

```
        IDictionary environmentVariables =
Environment.GetEnvironmentVariables();
        foreach (DictionaryEntry de in environmentVariables)
        {
            Console.WriteLine("  {0} = {1}", de.Key, de.Value);
        }

        Console.WriteLine("GetFolderPath: {0}",

Environment.GetFolderPath(Environment.SpecialFolder.System));

        string[] drives = Environment.GetLogicalDrives();
        Console.WriteLine("GetLogicalDrives: {0}", String.Join(", ",
drives));
    }
}
/*
This example produces results similar to the following:
(Any result that is lengthy or reveals information that should remain
secure has been omitted and marked "!---OMITTED---!".)

C:\>env0 ARBITRARY TEXT

-- Environment members --
CommandLine: env0 ARBITRARY TEXT
GetCommandLineArgs: env0, ARBITRARY, TEXT
CurrentDirectory: C:\Documents and Settings\!---OMITTED---!
ExitCode: 0
HasShutdownStarted: False
MachineName: !---OMITTED---!
NewLine:
  first line
  second line
  third line
OSVersion: Microsoft Windows NT 5.1.2600.0
StackTrace: '   at System.Environment.GetStackTrace(Exception e)
   at System.Environment.GetStackTrace(Exception e)
   at System.Environment.get_StackTrace()
   at Sample.Main()'
SystemDirectory: C:\WINNT\System32
TickCount: 17995355
UserDomainName: !---OMITTED---!
UserInteractive: True
UserName: !---OMITTED---!
Version: !---OMITTED---!
WorkingSet: 5038080
ExpandEnvironmentVariables:
  My system drive is C: and my system root is C:\WINNT
GetEnvironmentVariable:
  My temporary directory is C:\DOCUME~1\!---OMITTED---!\LOCALS~1\Temp.
GetEnvironmentVariables:
  !---OMITTED---!
GetFolderPath: C:\WINNT\System32
GetLogicalDrives: A:\, C:\, D:\
```

```
*/
```

Remarks

Use the Environment class to retrieve information such as command-line arguments, the exit code, environment variable settings, contents of the call stack, time since last system boot, and the version of the common language runtime.

Properties

Expand table

CommandLine	Gets the command line for this process.
CurrentDirectory	Gets or sets the fully qualified path of the current working directory.
CurrentManagedThreadId	Gets a unique identifier for the current managed thread.
ExitCode	Gets or sets the exit code of the process.
HasShutdownStarted	Gets a value that indicates whether the current application domain is being unloaded or the common language runtime (CLR) is shutting down.
Is64BitOperatingSystem	Gets a value that indicates whether the current operating system is a 64-bit operating system.
Is64BitProcess	Gets a value that indicates whether the current process is a 64-bit process.
IsPrivilegedProcess	Gets a value that indicates whether the current process is authorized to perform security-relevant functions.
MachineName	Gets the NetBIOS name of this local computer.
NewLine	Gets the newline string defined for this environment.
OSVersion	Gets the current platform identifier and version number.
ProcessId	Gets the unique identifier for the current process.
ProcessorCount	Gets the number of processors available to the current process.
ProcessPath	Returns the path of the executable that started the currently executing process. Returns `null` when the path is not available.
StackTrace	Gets current stack trace information.

SystemDirectory	Gets the fully qualified path of the system directory.
SystemPageSize	Gets the number of bytes in the operating system's memory page.
TickCount	Gets the number of milliseconds elapsed since the system started.
TickCount64	Gets the number of milliseconds elapsed since the system started.
UserDomainName	Gets the network domain name associated with the current user.
UserInteractive	Gets a value indicating whether the current process is running in user interactive mode.
UserName	Gets the user name of the person who is associated with the current thread.
Version	Gets a version consisting of the major, minor, build, and revision numbers of the common language runtime.
WorkingSet	Gets the amount of physical memory mapped to the process context.

Methods

Expand table

Exit(Int32)	Terminates this process and returns an exit code to the operating system.
ExpandEnvironment Variables(String)	Replaces the name of each environment variable embedded in the specified string with the string equivalent of the value of the variable, then returns the resulting string.
FailFast(String)	Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message is included in error reporting to Microsoft. For Unix-like systems, the message, alongside the stack trace, is written to the standard error stream.
FailFast(String, Exception)	Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message and exception information is included in error reporting to Microsoft. For Unix-like systems, the message alongside the stack trace is written to the standard error stream.
GetCommandLineArgs()	Returns a string array containing the command-line arguments for the current process.

GetEnvironmentVariable(String)	Retrieves the value of an environment variable from the current process.
GetEnvironmentVariable(String, EnvironmentVariableTarget)	Retrieves the value of an environment variable from the current process or from the Windows operating system registry key for the current user or local machine.
GetEnvironmentVariables()	Retrieves all environment variable names and their values from the current process.
GetEnvironment Variables(EnvironmentVariable Target)	Retrieves all environment variable names and their values from the current process, or from the Windows operating system registry key for the current user or local machine.
GetFolder Path(Environment+SpecialFolder)	Gets the path to the specified system special folder.
GetFolder Path(Environment+SpecialFolder, Environment+SpecialFolderOption)	Gets the path to the specified system special folder using a specified option for accessing special folders.
GetLogicalDrives()	Returns an array of string containing the names of the logical drives on the current computer.
SetEnvironmentVariable(String, String)	Creates, modifies, or deletes an environment variable stored in the current process.
SetEnvironmentVariable(String, String, EnvironmentVariableTarget)	Creates, modifies, or deletes an environment variable stored in the current process or in the Windows operating system registry key reserved for the current user or local machine.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Environment.CommandLine Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the command line for this process.

C#

```
public static string CommandLine { get; }
```

Property Value

String

A string containing command-line arguments.

Examples

The following example displays its own command line.

C#

```
using System;

class Example
{
    public static void Main()
    {
       Console.WriteLine();
       //  Invoke this sample with an arbitrary set of command line
arguments.
       Console.WriteLine("CommandLine: {0}", Environment.CommandLine);
    }
}
// The example displays output like the following:
//       C:\>env0 ARBITRARY TEXT
//
//       CommandLine: env0 ARBITRARY TEXT
```

Remarks

This property provides access to the program name and any arguments specified on the command line when the current process was started.

The program name can include path information, but is not required to do so. Use the GetCommandLineArgs method to retrieve the command-line information parsed and stored in an array of strings.

The maximum size of the command-line buffer is not set to a specific number of characters; it varies depending on the Windows operating system that is running on the computer.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetCommandLineArgs()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.CurrentDirectory Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the fully qualified path of the current working directory.

C#

```
public static string CurrentDirectory { get; set; }
```

Property Value

String

The directory path.

Exceptions

ArgumentException

Attempted to set to an empty string ("").

ArgumentNullException

Attempted to set to `null`.

IOException

An I/O error occurred.

DirectoryNotFoundException

Attempted to set a local path that cannot be found.

SecurityException

The caller does not have the appropriate permission.

Examples

The following example demonstrates setting the CurrentDirectory property.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      if (Environment.OSVersion.Platform == PlatformID.Win32NT)
      {
         // Change the directory to %WINDIR%
         Environment.CurrentDirectory =
Environment.GetEnvironmentVariable("windir");
         DirectoryInfo info = new DirectoryInfo(".");

         Console.WriteLine("Directory Info:   " + info.FullName);
      }
      else
      {
         Console.WriteLine("This example runs on Windows only.");
      }
   }
}
// The example displays output like the following on a .NET implementation
running on Windows:
//        Directory Info:   C:\windows
// The example displays the following output on a .NET implementation on
Unix-based systems:
//        This example runs on Windows only.
```

Remarks

By definition, if this process starts in the root directory of a local or network drive, the value of this property is the drive name followed by a trailing slash (for example, "C:\"). If this process starts in a subdirectory, the value of this property is the drive and subdirectory path, without a trailing slash (for example, "C:\mySubDirectory").

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.CurrentManagedThreadId Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a unique identifier for the current managed thread.

C#

```
public static int CurrentManagedThreadId { get; }
```

Property Value

Int32

A unique identifier for this managed thread.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Environment.ExitCode Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the exit code of the process.

C#

```
public static int ExitCode { get; set; }
```

Property Value

Int32

A 32-bit signed integer containing the exit code. The default value is 0 (zero), which indicates that the process completed successfully.

Examples

The following is a simple app named Double.exe that doubles an integer value passed to it as a command-line argument. The value assigns error codes to the ExitCode property to indicate error conditions. Note that you must add a reference to the System.Numerics.dll assembly to successfully compile the example.

C#

```
using System;
using System.Numerics;

public class Example
{
   private const int ERROR_BAD_ARGUMENTS = 0xA0;
   private const int ERROR_ARITHMETIC_OVERFLOW = 0x216;
   private const int ERROR_INVALID_COMMAND_LINE = 0x667;

   public static void Main()
   {
      string[] args = Environment.GetCommandLineArgs();
      if (args.Length == 1) {
```

```
            Environment.ExitCode = ERROR_INVALID_COMMAND_LINE;
        }
        else {
            BigInteger value = 0;
            if (BigInteger.TryParse(args[1], out value))
                if (value <= Int32.MinValue || value >= Int32.MaxValue)
                    Environment.ExitCode = ERROR_ARITHMETIC_OVERFLOW;
                else
                    Console.WriteLine("Result: {0}", value * 2);

            else
                Environment.ExitCode = ERROR_BAD_ARGUMENTS;
        }
    }
}
```

The example can then be invoked from a batch file such as the following, which makes its error codes accessible by using the `ERRORLEVEL` command.

```
@echo off
Double.exe %1
if errorlevel 1639 goto NoArg
if errorlevel 534 goto Overflow
if errorlevel 160 goto BadArg
if errorlevel 0 echo Completed Successfully
goto :EOF

:NoArg
echo Missing argument
goto :EOF

:Overflow
echo Arithmetic overflow
goto :EOF

:BadArg
echo Invalid argument
goto :EOF
```

The following shows some sample output produced by invoking the batch file.

```
>getdouble 123
Result: 246
Completed Successfully

>getdouble 5912323109093
Arithmetic overflow
```

```
>getdouble
Missing argument

>getdouble "a string"
Invalid argument
```

Note that code for Double.exe is identical in function to the following example, except that the former defines an entry point named `Main` that has no return value, whereas this example defines an entry point named `Main` that returns an integer.

C#

```
using System;
using System.Numerics;

public class Example
{
   private const int ERROR_SUCCESS = 0;
   private const int ERROR_BAD_ARGUMENTS = 0xA0;
   private const int ERROR_ARITHMETIC_OVERFLOW = 0x216;
   private const int ERROR_INVALID_COMMAND_LINE = 0x667;

   public static int Main()
   {
      string[] args = Environment.GetCommandLineArgs();
      if (args.Length == 1) {
         return ERROR_INVALID_COMMAND_LINE;
      }
      else {
         BigInteger value = 0;
         if (BigInteger.TryParse(args[1], out value))
            if (value <= Int32.MinValue || value >= Int32.MaxValue)
               return ERROR_ARITHMETIC_OVERFLOW;
            else
               Console.WriteLine("Result: {0}", value * 2);

         else
            return ERROR_BAD_ARGUMENTS;
      }
      return ERROR_SUCCESS;
   }
}
```

Remarks

If the `Main` method returns `void`, you can use this property to set the exit code that will be returned to the calling environment. If `Main` does not return `void`, this property is ignored. The initial value of this property is zero.

> ⚠ **Warning**
>
> The **ExitCode** property is a signed 32-bit integer. To prevent the property from returning a negative exit code, you should not use values greater than or equal to 0x80000000.

Use a non-zero number to indicate an error. In your application, you can define your own error codes in an enumeration, and return the appropriate error code based on the scenario. For example, return a value of 1 to indicate that the required file is not present and a value of 2 to indicate that the file is in the wrong format. For a list of exit codes used by the Windows operating system, see System Error Codes in the Windows documentation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.HasShutdownStarted Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current application domain is being unloaded or the common language runtime (CLR) is shutting down.

```
C#

public static bool HasShutdownStarted { get; }
```

Property Value

Boolean

`true` if the current application domain is being unloaded or the CLR is shutting down; otherwise, `false`.

Remarks

.NET Framework only: When the CLR unloads an application domain, it runs the finalizers on all objects that have a finalizer method in that application domain. When the CLR shuts down, it starts the finalizer thread on all objects that have a finalizer method. The HasShutdownStarted property returns `true` only after the finalizer thread has been started. When the property returns `true`, you can determine whether an application domain is being unloaded or the CLR itself is shutting down by calling the AppDomain.IsFinalizingForUnload method. This method returns `true` if finalizers are called because the application domain is unloading or `false` if the CLR is shutting down.

The HasShutdownStarted property returns `false` if the finalizer thread has not been started.

By using this property, you can determine whether to access static variables in your finalization code. If either an application domain or the CLR is shutting down, you cannot reliably access any object that has a finalization method and that is referenced by a static field. This is because these objects may have already been finalized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsFinalizingForUnload()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.Is64BitOperatingSystem Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current operating system is a 64-bit operating system.

C#

```
public static bool Is64BitOperatingSystem { get; }
```

Property Value

Boolean

`true` if the operating system is 64-bit; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Environment.Is64BitProcess Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current process is a 64-bit process.

C#

```
public static bool Is64BitProcess { get; }
```

Property Value

Boolean

`true` if the process is 64-bit; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.IsPrivilegedProcess Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current process is authorized to perform security-relevant functions.

C#

```
public static bool IsPrivilegedProcess { get; }
```

Property Value

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.MachineName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the NetBIOS name of this local computer.

C#

```
public static string MachineName { get; }
```

Property Value

String

The name of this computer.

Exceptions

InvalidOperationException

The name of this computer cannot be obtained.

Examples

The following example displays the name of the computer that runs the code example. (The machine name is omitted from the example output for security reasons.)

C#

```
// Sample for the Environment.MachineName property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
//  <-- Keep this information secure! -->
    Console.WriteLine("MachineName: {0}", Environment.MachineName);
    }
```

```
}
/*
This example produces the following results:
(Any result that is lengthy, specific to the machine on which this sample
was tested, or reveals information that should remain secure, has been
omitted and marked "!---OMITTED---!".)

MachineName: !---OMITTED---!
*/
```

Remarks

The name of this computer is established at system startup when the name is read from the registry. If this computer is a node in a cluster, the name of the node is returned.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.NewLine Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the newline string defined for this environment.

C#

```
public static string NewLine { get; }
```

Property Value

String

`\r\n` for non-Unix platforms, or `\n` for Unix platforms.

Examples

The following example displays two lines separated by a newline.

C#

```
// Sample for the Environment.NewLine property
using System;

class Sample
{
    public static void Main()
    {
        Console.WriteLine();
        Console.WriteLine($"NewLine: {Environment.NewLine}  first
line{Environment.NewLine}  second line");
    }
}

/*
This example produces the following results:

NewLine:
  first line
```

```
  second line
*/
```

Remarks

The property value of NewLine is a constant customized specifically for the current platform and implementation of the .NET Framework. For more information about the escape characters in the property value, see Character Escapes.

The functionality provided by NewLine is often what is meant by the terms newline, line feed, line break, carriage return, CRLF, and end of line.

NewLine can be used in conjunction with language-specific newline support such as the escape characters '\r' and '\n' in Microsoft C# and C/C++, or `vbCrLf` in Microsoft Visual Basic.

NewLine is automatically appended to text processed by the Console.WriteLine and StringBuilder.AppendLine methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.OSVersion Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current platform identifier and version number.

```
C#
public static OperatingSystem OSVersion { get; }
```

Property Value

OperatingSystem

The platform identifier and version number.

Exceptions

InvalidOperationException

This property was unable to obtain the system version.

-or-

The obtained platform identifier is not a member of PlatformID

Remarks

Environment.OSVersion returns an OS version that may be incorrect when an application runs under Windows compatibility mode. For more information, see GetVersionExA function remarks.

In .NET 5 and later versions, Environment.OSVersion returns the actual OS version in compatibility modes specified by the application manifest. For more information, see Environment.OSVersion returns the correct operating system version.

To identify the operating system platform, for example, Linux or Windows, you can use the RuntimeInformation.IsOSPlatform method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.ProcessId Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the unique identifier for the current process.

C#

```
public static int ProcessId { get; }
```

Property Value

Int32

A number that represents the unique identifier for the current process.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.ProcessorCount Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of processors available to the current process.

C#

```
public static int ProcessorCount { get; }
```

Property Value

Int32

The 32-bit signed integer that specifies the number of processors that are available.

Examples

The following example demonstrates the ProcessorCount property.

C#

```
// This example demonstrates the
//     Environment.ProcessorCount property.
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine("The number of processors " +
        "on this computer is {0}.",
        Environment.ProcessorCount);
    }
}
/*
This example produces the following results:

The number of processors on this computer is 1.
*/
```

Remarks

On Linux and macOS systems for all .NET versions and on Windows systems starting with .NET 6, this API returns the minimum of:

- The number of logical processors on the machine.
- If the process is running with CPU affinity, the number of processors that the process is affinitized to.
- If the process is running with a CPU utilization limit, the CPU utilization limit rounded up to the next whole number.

The value returned by this API is fixed at .NET runtime startup for the process lifetime. It does not reflect changes in the environment settings while the process is running.

For more information about processor groups and logical processors, see Processor Groups.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.ProcessPath Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the path of the executable that started the currently executing process. Returns `null` when the path is not available.

```
C#
public static string? ProcessPath { get; }
```

Property Value

String

The path of the executable that started the currently executing process.

Remarks

If the executable is renamed or deleted before this property is first accessed, the return value is undefined and depends on the operating system.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Environment.StackTrace Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets current stack trace information.

C#

```
public static string StackTrace { get; }
```

Property Value

String

A string containing stack trace information. This value can be Empty.

Examples

The following example demonstrates the StackTrace property.

C#

```
// Sample for the Environment.StackTrace property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    Console.WriteLine("StackTrace: '{0}'", Environment.StackTrace);
    }
}
/*
This example produces the following results:

StackTrace: '   at System.Environment.GetStackTrace(Exception e)
   at System.Environment.GetStackTrace(Exception e)
   at System.Environment.get_StackTrace()
```

```
    at Sample.Main()'
 */
```

Remarks

The StackTrace property lists method calls in reverse chronological order, that is, the most recent method call is described first, and one line of stack trace information is listed for each method call on the stack. However, the StackTrace property might not report as many method calls as expected due to code transformations that occur during optimization.

> **Note**
>
> For a hierarchical view of the stack trace information by class, use the **StackTrace** class.

The StackTrace property formats the stack trace information for each method call as follows:

"at `FullClassName`.`MethodName`(`MethodParams`) in `FileName` :line `LineNumber` "

The literal "at" is preceded by three spaces, and the entire substring starting with "in" is omitted if debug symbols are not available. The placeholders, `FullClassName`, `MethodName`, `MethodParams`, `FileName`, and `LineNumber`, are replaced by actual values, and are defined as follows:

`FullClassName` The full name of the class, including the namespace.

`MethodName` The name of the method.

`MethodParams` The list of parameter type/name pairs. Each pair is separated by a comma (","). This information is omitted if `MethodName` takes no parameters.

`FileName` The name of the source file where the `MethodName` method is declared. This information is omitted if debug symbols are not available.

`LineNumber` The number of the line in `FileName` that contains the source code from `MethodName` for the instruction that is on the call stack. This information is omitted if debug symbols are not available.

The Environment.NewLine string terminates each line of the stack trace.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.SystemDirectory Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fully qualified path of the system directory.

C#

```
public static string SystemDirectory { get; }
```

Property Value

String

A string containing a directory path.

Examples

The following example displays the system directory of the computer that runs the code example. (The system directory is omitted from the example output for security reasons.)

C#

```
// Sample for the Environment.SystemDirectory property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
//  <-- Keep this information secure! -->
    Console.WriteLine("SystemDirectory: {0}", Environment.SystemDirectory);
    }
}
/*
This example produces the following results:
```

```
SystemDirectory: C:\WINNT\System32
*/
```

Remarks

An example of the value returned is the string "C:\WINDOWS\System32".

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.SystemPageSize Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes in the operating system's memory page.

C#

```
public static int SystemPageSize { get; }
```

Property Value

Int32

The number of bytes in the system memory page.

Remarks

This information can be useful when determining whether to use the MemoryMappedFileOptions.DelayAllocatePages option when you work with memory-mapped files.

In Windows, this value is the `dwPageSize` member in the `SYSTEM_INFO` structure.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Environment.TickCount Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of milliseconds elapsed since the system started.

C#

```
public static int TickCount { get; }
```

Property Value

Int32

A 32-bit signed integer containing the amount of time in milliseconds that has passed since the last time the computer was started.

Examples

The following example demonstrates how to retrieve the positive range of values returned by the TickCount property. The TickCount property cycles between Int32.MinValue, which is a negative number, and Int32.MaxValue once every 49.8 days. This code sample removes the sign bit to yield a nonnegative number that cycles between zero and MaxValue once every 24.9 days.

C#

```
// Sample for the Environment.TickCount property.

// TickCount cycles between Int32.MinValue, which is a negative
// number, and Int32.MaxValue once every 49.8 days. This sample
// removes the sign bit to yield a nonnegative number that cycles
// between zero and Int32.MaxValue once every 24.9 days.

using System;

class Sample
{
    public static void Main()
```

```
    {
    int result = Environment.TickCount & Int32.MaxValue;
    Console.WriteLine("TickCount: {0}", result);
    }
}
/*
This example produces an output similar to the following:

TickCount: 101931139
*/
```

Remarks

The value of this property is derived from the system timer and is stored as a 32-bit signed integer. Note that, because it is derived from the system timer, the resolution of the TickCount property is limited to the resolution of the system timer, which is typically in the range of 10 to 16 milliseconds.

> ⓘ **Important**
>
> Because the value of the **TickCount** property value is a 32-bit signed integer, if the system runs continuously, **TickCount** will increment from zero to **Int32.MaxValue** for approximately 24.9 days, then jump to **Int32.MinValue**, which is a negative number, then increment back to zero during the next 24.9 days. You can work around this issue by calling the Windows **GetTickCount** function, which resets to zero after approximately 49.7 days, or by calling the **GetTickCount64** function.

TickCount is different from the DateTime.Ticks property, which is the number of 100-nanosecond intervals that have elapsed since 1/1/0001, 12:00am.

Use the DateTime.Now property to obtain the current local date and time on this computer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Environment.TickCount64 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of milliseconds elapsed since the system started.

C#

```
public static long TickCount64 { get; }
```

Property Value

Int64

The elapsed milliseconds since the system started.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.UserDomainName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the network domain name associated with the current user.

```
C#
public static string UserDomainName { get; }
```

Property Value

String

The network domain name associated with the current user.

Exceptions

PlatformNotSupportedException

The operating system does not support retrieving the network domain name.

InvalidOperationException

The network domain name cannot be retrieved.

Remarks

The domain account credentials for a user are formatted as the user's domain name, the '\' character, and user name. Use the UserDomainName property to obtain the user's domain name without the user name, and the UserName property to obtain the user name without the domain name. For example, if a user's domain name and user name are CORPORATENETWORK\john, the UserDomainName property returns "CORPORATENETWORK".

The UserDomainName property first attempts to get the domain name component of the Windows account name for the current user. If that attempt fails, this property attempts to get the domain name associated with the user name provided by the UserName property. If that attempt fails because the host computer is not joined to a domain, then the host computer name is returned.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- UserName

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.UserInteractive Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current process is running in user interactive mode.

C#

```
public static bool UserInteractive { get; }
```

Property Value

Boolean

`true` if the current process is running in user interactive mode; otherwise, `false`.

Examples

The following example displays whether the current process is running in user interactive mode.

C#

```
// Sample for the Environment.UserInteractive property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    Console.WriteLine("UserInteractive: {0}", Environment.UserInteractive);
    }
}
/*
This example produces the following results:

UserInteractive: True
*/
```

Remarks

The UserInteractive property reports `false` for a Windows process or a service like IIS that runs without a user interface. If this property is `false`, do not display modal dialogs or message boxes because there is no graphical user interface for the user to interact with.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SystemInformation

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.UserName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the user name of the person who is associated with the current thread.

C#

```
public static string UserName { get; }
```

Property Value

String

The user name of the person who is associated with the current thread.

Examples

The following example displays the user name of the person who started the current thread.

C#

```
// Sample for the Environment.UserName property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
//  <-- Keep this information secure! -->
    Console.WriteLine("UserName: {0}", Environment.UserName);
    }
}
```

Remarks

You can use the UserName property to identify the user on the current thread, to the system and application for security or access purposes. It can also be used to customize a particular application for each user.

On Windows the UserName property wraps a call to the Windows GetUserName function. The domain account credentials for a user are formatted as the user's domain name, the '\' character, and user name. Use the UserDomainName property to obtain the user's domain name and the UserName property to obtain the user name.

On Unix platforms the UserName property wraps a call to the `getpwuid_r` function.

If an ASP.NET application runs in a development environment, the UserName property returns the name of the current user. In a published ASP.NET application, this property returns the name of the application pool account (such as Default AppPool).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- UserDomainName

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.Version Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a version consisting of the major, minor, build, and revision numbers of the common language runtime.

```
C#

public static Version Version { get; }
```

Property Value

Version

The version of the common language runtime.

Examples

The following example displays the version of the common language runtime.

```
C#

// Sample for the Environment.Version property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    Console.WriteLine("Version: {0}", Environment.Version.ToString());
    }
}
```

Remarks

For .NET Core 3.x and .NET 5+, the Environment.Version property returns the .NET runtime version number.

For the .NET Framework Versions 4, 4.5, 4.5.1, and 4.5.2, the Environment.Version property returns a Version object whose string representation has the form `4.0.30319.xxxxx`. For the .NET Framework 4.6 and later versions, and .NET Core versions before 3.0, it has the form `4.0.30319.42000`.

> ⚠ **Warning**
>
> For the .NET Framework 4.5 and later, we do not recommend using the **Version** property to detect the version of the runtime; instead, you can determine the version of the common language runtime by querying the registry. For more information, see **How to: Determine Which .NET Framework Versions Are Installed**.

For more information about the version of the common language runtime that is installed with each version of the .NET Framework, see Versions and Dependencies.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.WorkingSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the amount of physical memory mapped to the process context.

C#

```
public static long WorkingSet { get; }
```

Property Value

Int64

A 64-bit signed integer containing the number of bytes of physical memory mapped to the process context.

Examples

The following example displays the size of the working set of the computer that runs the code example.

C#

```
// Sample for the Environment.WorkingSet property
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine("WorkingSet: {0}", Environment.WorkingSet);
    }
}
/*
This example produces the following results:

WorkingSet: 5038080
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.Exit(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Terminates this process and returns an exit code to the operating system.

```
C#
public static void Exit (int exitCode);
```

Parameters

`exitCode` Int32

The exit code to return to the operating system. Use 0 (zero) to indicate that the process completed successfully.

Exceptions

SecurityException

The caller does not have sufficient security permission to perform this function.

Remarks

For the `exitCode` parameter, use a non-zero number to indicate an error. In your application, you can define your own error codes in an enumeration, and return the appropriate error code based on the scenario. For example, return a value of 1 to indicate that the required file is not present, and a value of 2 to indicate that the file is in the wrong format. For a list of exit codes used by the Windows operating system, see System Error Codes in the Windows documentation.

Calling the Exit method differs from using your programming language's `return` statement in the following ways:

- Exit always terminates an application. Using the `return` statement may terminate an application only if it is used in the application entry point, such as in the `Main` method.

- Exit terminates an application immediately, even if other threads are running. If the `return` statement is called in the application entry point, it causes an application to terminate only after all foreground threads have terminated.
- Exit requires the caller to have permission to call unmanaged code. The `return` statement does not.
- If Exit is called from a `try` or `catch` block, the code in any `finally` block does not execute. If the `return` statement is used, the code in the `finally` block does execute.
- If Exit is called when code in a constrained execution region (CER) is running, the CER will not complete execution. If the `return` statement is used, the CER completes execution.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.ExpandEnvironment Variables(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Replaces the name of each environment variable embedded in the specified string with the string equivalent of the value of the variable, then returns the resulting string.

C#

```
public static string ExpandEnvironmentVariables (string name);
```

Parameters

name String

A string containing the names of zero or more environment variables. Each environment variable is quoted with the percent sign character (%).

Returns

String

A string with each environment variable replaced by its value.

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following example shows how to obtain the system drive and system root variables.

C#

```
// Sample for the Environment.ExpandEnvironmentVariables method
using System;

class Sample
{
    public static void Main()
    {
        // Keep this information secure!
        string query = "My system drive is %SystemDrive% and my system root
is %SystemRoot%";

        string str = Environment.ExpandEnvironmentVariables(query);

        Console.WriteLine(str);
    }
}
/*
This example prints:

My system drive is C: and my system root is C:\WINDOWS
*/
```

Remarks

COM interop is used to retrieve the environment variables from the operating system. If the environment variables cannot be retrieved due to a COM error, the HRESULT that explains the cause of the failure is used to generate one of several possible exceptions; that is, the exception depends on the HRESULT. For more information about how the HRESULT is processed, see the Remarks section of the Marshal.ThrowExceptionForHR method.

Replacement only occurs for environment variables that are set. For example, suppose `name` is "MyENV = %MyENV%". If the environment variable, MyENV, is set to 42, this method returns "MyENV = 42". If MyENV is not set, no change occurs; this method returns "MyENV = %MyENV%".

The size of the return value is limited to 32K.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.FailFast Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message and optional exception information is included in error reporting to Microsoft. For Unix-like systems, the message, alongside the stack trace, is written to the standard error stream.

Overloads

Expand table

FailFast(String)	Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message is included in error reporting to Microsoft. For Unix-like systems, the message, alongside the stack trace, is written to the standard error stream.
FailFast(String, Exception)	Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message and exception information is included in error reporting to Microsoft. For Unix-like systems, the message alongside the stack trace is written to the standard error stream.

FailFast(String)

Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message is included in error reporting to Microsoft. For Unix-like systems, the message, alongside the stack trace, is written to the standard error stream.

C#

```
public static void FailFast (string? message);
```

Parameters

`message` String

A message that explains why the process was terminated, or `null` if no explanation is provided.

Examples

The following example writes a log entry to the Windows Application event log when running under Windows, or writes the error message to the standard error stream when running under a Unix-like system, and terminates the current process.

C#

```
using System;

class Example
{
    public static void Main()
    {
       string causeOfFailure = "A catastrophic failure has occurred.";

       // Assume your application has failed catastrophically and must
       // terminate immediately. The try-finally block is not executed
       // and is included only to demonstrate that instructions within
       // try-catch blocks and finalizers are not performed.
       try
       {
           Environment.FailFast(causeOfFailure);
       }
       finally
       {
           Console.WriteLine("This finally block will not be executed.");
       }
   }
}
/*
The example produces no output because the application is terminated.
However, an entry is made in the Windows Application event log, and
the log entry contains the text from the causeOfFailure variable.
*/
```

Remarks

This method terminates a process without running any active `try`/`finally` blocks or finalizers.

On Windows, the `Environment.FailFast` method writes the `message` string to the Windows Application event log, creates a dump of your application, and then terminates the current process. The `message` string is also included in error reporting to Microsoft via Windows Error Reporting. For more information, see Windows Error Reporting: Getting Started.

On Unix-like systems, the message is written to the standard error stream, alongside the stack trace information.

Use the `Environment.FailFast` method instead of the Exit method to terminate your application if the state of your application is damaged beyond repair, and executing your application's `try`/`finally` blocks and finalizers will corrupt program resources.

Calling the `Environment.FailFast` method to terminate execution of an application running in the Visual Studio debugger throws an ExecutionEngineException and automatically triggers the fatalExecutionEngineError managed debugging assistant (MDA).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FailFast(String, Exception)

Immediately terminates the process before reporting an error message. For Windows, the error message is written to the Windows Application event log, and the message and exception information is included in error reporting to Microsoft. For Unix-like systems, the message alongside the stack trace is written to the standard error stream.

```
C#
```

```
public static void FailFast (string? message, Exception? exception);
```

Parameters

`message` String

A message that explains why the process was terminated, or `null` if no explanation is provided.

`exception` Exception

An exception that represents the error that caused the termination. This is typically the exception in a `catch` block.

Remarks

This method terminates the process without running any active `try`/`finally` blocks or finalizers.

On Windows, the `Environment.FailFast` method writes the `message` string to the Windows Application event log, creates a dump of your application, and then terminates the current process.

Information is reported to Microsoft by using Windows Error Reporting. For more information, see Windows Error Reporting: Getting Started. Error reporting to Microsoft includes `message` and `exception` information, which provides details used to classify the error. Although `exception` is not handled because the process is terminated, the contextual information that raised the exception is still obtained.

On Unix-like systems, the message is written to the standard error stream, alongside the stack trace information.

If `exception` is `null`, or if `exception` is not thrown, this method operates the same as the FailFast(String) method overload.

Use the `Environment.FailFast` method instead of the Exit method to terminate your application if the state of your application is damaged beyond repair, and executing your application's `try`/`finally` blocks and finalizers will corrupt program resources.

Calling the `Environment.FailFast` method to terminate execution of an application running in the Visual Studio debugger throws an ExecutionEngineException and automatically triggers the fatalExecutionEngineError managed debugging assistant (MDA).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.GetCommandLineArgs Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string array containing the command-line arguments for the current process.

C#

```
public static string[] GetCommandLineArgs ();
```

Returns

String[]

An array of strings where each element contains a command-line argument. The first element is the executable file name, and the following zero or more elements contain the remaining command-line arguments.

Exceptions

NotSupportedException

The system does not support command-line arguments.

Examples

The following example displays the application's command line arguments.

C#

```
using System;

class Sample
{
    public static void Main()
    {
        Console.WriteLine();
```

```
        //  Invoke this sample with an arbitrary set of command line
arguments.
        string[] arguments = Environment.GetCommandLineArgs();
        Console.WriteLine("GetCommandLineArgs: {0}", string.Join(", ",
arguments));
    }
}
/*
This example produces output like the following:

    C:\>GetCommandLineArgs ARBITRARY TEXT

      GetCommandLineArgs: GetCommandLineArgs, ARBITRARY, TEXT
*/
```

Remarks

The first element in the array contains the file name of the executing program. If the file name is not available, the first element is equal to String.Empty. The remaining elements contain any additional tokens entered on the command line.

In .NET 5 and later versions, for single-file publishing, the first element is the name of the host executable.

The program file name can, but is not required to, include path information.

Command line arguments are delimited by spaces. You can use double quotation marks (") to include spaces within an argument. The single quotation mark (), however, does not provide this functionality.

If a double quotation mark follows two or an even number of backslashes, each proceeding backslash pair is replaced with one backslash and the double quotation mark is removed. If a double quotation mark follows an odd number of backslashes, including just one, each preceding pair is replaced with one backslash and the remaining backslash is removed; however, in this case the double quotation mark is not removed.

The following table shows how command line arguments can be delimited, and assumes `MyApp` as the current executing application.

Expand table

Input at the command line	Resulting command line arguments
`MyApp alpha beta`	`MyApp, alpha, beta`

Input at the command line	Resulting command line arguments
`MyApp "alpha with spaces" "beta with spaces"`	`MyApp, alpha with spaces, beta with spaces`
`MyApp 'alpha with spaces' beta`	`MyApp, 'alpha, with, spaces', beta`
`MyApp \\\alpha \\\\"beta`	`MyApp, \\\alpha, \\beta`
`MyApp \\\\\"alpha \"beta`	`MyApp, \\"alpha, "beta`

To obtain the command line as a single string, use the CommandLine property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

See also

- CommandLine

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.GetEnvironmentVariable Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the value of an environment variable.

Overloads

Expand table

GetEnvironmentVariable(String)	Retrieves the value of an environment variable from the current process.
GetEnvironmentVariable(String, EnvironmentVariableTarget)	Retrieves the value of an environment variable from the current process or from the Windows operating system registry key for the current user or local machine.

GetEnvironmentVariable(String)

Retrieves the value of an environment variable from the current process.

C#

```
public static string? GetEnvironmentVariable (string variable);
```

Parameters

`variable` String

The name of the environment variable.

Returns

String

The value of the environment variable specified by `variable`, or `null` if the environment variable is not found.

Exceptions

ArgumentNullException

`variable` is `null`.

SecurityException

The caller does not have the required permission to perform this operation.

Examples

The following example uses the GetEnvironmentVariable method to retrieve the `windir` environment variable, which contains the path of the Windows directory.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      if (Environment.OSVersion.Platform == PlatformID.Win32NT)
      {
         // Change the directory to %WINDIR%
         Environment.CurrentDirectory =
Environment.GetEnvironmentVariable("windir");
         DirectoryInfo info = new DirectoryInfo(".");

         Console.WriteLine("Directory Info:   " + info.FullName);
      }
      else
      {
         Console.WriteLine("This example runs on Windows only.");
      }
   }
}
// The example displays output like the following on a .NET
implementation running on Windows:
//        Directory Info:   C:\windows
// The example displays the following output on a .NET implementation on
Unix-based systems:
//        This example runs on Windows only.
```

The following example attempts to retrieve the value of an environment variable named `Test1` from the process environment block. If the variable doesn't exist, the example creates it and retrieves its value. The example displays the value of the variable. If the example created the variable, it also calls the GetEnvironmentVariables(EnvironmentVariableTarget) method with each member of the EnvironmentVariableTarget enumeration to establish that the variable can be retrieved only from the current process environment block. Finally, if the example created the variable, it deletes it.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string value;
      bool toDelete = false;

      // Check whether the environment variable exists.
      value = Environment.GetEnvironmentVariable("Test1");
      // If necessary, create it.
      if (value == null)
      {
         Environment.SetEnvironmentVariable("Test1", "Value1");
         toDelete = true;

         // Now retrieve it.
         value = Environment.GetEnvironmentVariable("Test1");
      }
      // Display the value.
      Console.WriteLine($"Test1: {value}\n");

      // Confirm that the value can only be retrieved from the process
      // environment block if running on a Windows system.
      if (Environment.OSVersion.Platform == PlatformID.Win32NT)
      {
         Console.WriteLine("Attempting to retrieve Test1 from:");
         foreach (EnvironmentVariableTarget enumValue in

Enum.GetValues(typeof(EnvironmentVariableTarget))) {
            value = Environment.GetEnvironmentVariable("Test1",
enumValue);
            Console.WriteLine($"   {enumValue}: {(value != null ? "found"
: "not found")}");
         }
         Console.WriteLine();
      }

      // If we've created it, now delete it.
```

```
        if (toDelete) {
           Environment.SetEnvironmentVariable("Test1", null);
           // Confirm the deletion.
           if (Environment.GetEnvironmentVariable("Test1") == null)
              Console.WriteLine("Test1 has been deleted.");
        }
    }
 }
 // The example displays the following output if run on a Windows system:
 //      Test1: Value1
 //
 //      Attempting to retrieve Test1 from:
 //         Process: found
 //         User: not found
 //         Machine: not found
 //
 //      Test1 has been deleted.
 //
 // The example displays the following output if run on a Unix-based
 system:
 //      Test1: Value1
 //
 //      Test1 has been deleted.
```

Remarks

For more information about this API, see Supplemental API remarks for Environment.GetEnvironmentVariable.

See also

- GetEnvironmentVariables()
- SetEnvironmentVariable(String, String)
- EnvironmentVariableTarget

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetEnvironmentVariable(String, EnvironmentVariableTarget)

Retrieves the value of an environment variable from the current process or from the Windows operating system registry key for the current user or local machine.

C#

```
public static string? GetEnvironmentVariable (string variable,
EnvironmentVariableTarget target);
```

Parameters

`variable` String

The name of an environment variable.

`target` EnvironmentVariableTarget

One of the EnvironmentVariableTarget values. Only Process is supported on .NET running on Unix-based systems.

Returns

String

The value of the environment variable specified by the `variable` and `target` parameters, or `null` if the environment variable is not found.

Exceptions

ArgumentNullException

`variable` is `null`.

ArgumentException

`target` is not a valid EnvironmentVariableTarget value.

SecurityException

The caller does not have the required permission to perform this operation.

Examples

The following example creates environment variables for the EnvironmentVariableTarget.Process, EnvironmentVariableTarget.User, and Machine targets, checks whether the operating system registry contains the user and machine environment variables, then deletes the environment variables. Because .NET on Unix-based systems does not support per-user and per-machine environment variables, only SetEnvironmentVariable(String, String) and SetEnvironmentVariable(String, String, EnvironmentVariableTarget) with a value of EnvironmentVariableTarget.Process successfully store an environment variable to the process environment block.

C#

```
using System;
using System.Collections;
using Microsoft.Win32;

class Sample
{
    public static void Main()
    {
        // Environment variable names for default, process, user, and
machine targets.
        string defaultEnvVar = nameof(defaultEnvVar);
        string processEnvVar = nameof(processEnvVar);
        string userEnvVar = nameof(userEnvVar);
        string machineEnvVar = nameof(machineEnvVar);

        string dft = nameof(dft);
        string process = nameof(process);
        string user = nameof(user);
        string machine = nameof(machine);

        // Set the environment variable for each target.
        Console.WriteLine("Setting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, dft);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, process,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, user,
```

```
EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, machine,

EnvironmentVariableTarget.Machine);

        // Define an array of environment variables.
        string[] envVars = { defaultEnvVar,processEnvVar, userEnvVar,
machineEnvVar };

        // Try to get the environment variables from each target.
        // The default (no specified target).
        Console.WriteLine("Retrieving environment variables from the
default target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar) ?? "
(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The process block.
        Console.WriteLine("\nRetrieving environment variables from the
Process target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Process) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The user block.
        Console.WriteLine("\nRetrieving environment variables from the
User target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.User) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The machine block.
        Console.WriteLine("\nRetrieving environment variables from the
Machine target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Machine) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }

        // Delete the environment variable for each target.
        Console.WriteLine("\nDeleting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, null);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, null,
```

```
EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, null,

EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, null,

EnvironmentVariableTarget.Machine);
    }
}
// The example displays the following output if run on a Windows system:
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: machine
//
//      Deleting environment variables for each target...
//
// The example displays the following output if run on a Unix-based
system:
//
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
```

```
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Deleting environment variables for each target...
```

Remarks

For more information about this API, see Supplemental API remarks for Environment.GetEnvironmentVariable.

See also

- SetEnvironmentVariable(String, String, EnvironmentVariableTarget)
- GetEnvironmentVariables()
- EnvironmentVariableTarget

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Environment.GetEnvironmentVariables Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves all environment variable names and their values.

Overloads

Expand table

GetEnvironmentVariables()	Retrieves all environment variable names and their values from the current process.
GetEnvironmentVariables(EnvironmentVariableTarget)	Retrieves all environment variable names and their values from the current process, or from the Windows operating system registry key for the current user or local machine.

GetEnvironmentVariables()

Retrieves all environment variable names and their values from the current process.

C#

```
public static System.Collections.IDictionary GetEnvironmentVariables ();
```

Returns

IDictionary

A dictionary that contains all environment variable names and their values; otherwise, an empty dictionary if no environment variables are found.

Exceptions

SecurityException

The caller does not have the required permission to perform this operation.

OutOfMemoryException

The buffer is out of memory.

Examples

The following example demonstrates the GetEnvironmentVariables method.

C#

```
// Sample for the Environment.GetEnvironmentVariables method
using System;
using System.Collections;

class Sample
{
    public static void Main()
    {
       Console.WriteLine();
       Console.WriteLine("GetEnvironmentVariables: ");
       foreach (DictionaryEntry de in
Environment.GetEnvironmentVariables())
           Console.WriteLine("  {0} = {1}", de.Key, de.Value);
    }
}
// Output from the example is not shown, since it is:
//    Lengthy.
//    Specific to the machine on which the example is run.
//    May reveal information that should remain secure.
```

Remarks

The names and values for the environment variables are stored as key-value pairs in the returned IDictionary.

On Windows systems

On Windows systems, the `GetEnvironmentVariables` method returns the following environment variables:

- All per-machine environment variables that are defined at the time the process is created, along with their values.

- All per-user environment variables that are defined at the time the process is created, along with their values.
- Any variables inherited from the parent process from which the .NET application was launched or added to the process block while the process is running. Environment variables are added while the process is running by calling either the SetEnvironmentVariable(String, String) method or the SetEnvironmentVariable(String, String, EnvironmentVariableTarget) method with a `target` value of EnvironmentVariableTarget.Process.

On non-Windows systems

On non-Windows systems, the `GetEnvironmentVariables` method retrieves the name and value of all environment variables that are inherited from the parent process that launched the `dotnet` process or that are defined within the scope of the `dotnet` process itself. Once the `dotnet` process ends, these latter environment variables cease to exist.

.NET running on Unix-based systems does not support per-machine or per-user environment variables.

See also

- GetEnvironmentVariable(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetEnvironmentVariables(EnvironmentVariableTarget)

Retrieves all environment variable names and their values from the current process, or from the Windows operating system registry key for the current user or local machine.

```
C#

public static System.Collections.IDictionary GetEnvironmentVariables
(EnvironmentVariableTarget target);
```

Parameters

`target` EnvironmentVariableTarget

One of the EnvironmentVariableTarget values. Only Process is supported on .NET running on Unix-based systems.

Returns

IDictionary

A dictionary that contains all environment variable names and their values from the source specified by the `target` parameter; otherwise, an empty dictionary if no environment variables are found.

Exceptions

SecurityException

The caller does not have the required permission to perform this operation for the specified value of `target`.

ArgumentException

`target` contains an illegal value.

Examples

The following example creates environment variables for the EnvironmentVariableTarget.Process, EnvironmentVariableTarget.User, and Machine targets, checks whether the operating system registry contains the user and machine environment variables, then deletes the environment variables. Because .NET on

Unix-based systems does not support per-user and per-machine environment variables, only SetEnvironmentVariable(String, String) and SetEnvironmentVariable(String, String, EnvironmentVariableTarget) with a value of EnvironmentVariableTarget.Process successfully store an environment variable to the process environment block.

C#

```
using System;
using System.Collections;
using Microsoft.Win32;

class Sample
{
    public static void Main()
    {
        // Environment variable names for default, process, user, and
machine targets.
        string defaultEnvVar = nameof(defaultEnvVar);
        string processEnvVar = nameof(processEnvVar);
        string userEnvVar = nameof(userEnvVar);
        string machineEnvVar = nameof(machineEnvVar);

        string dft = nameof(dft);
        string process = nameof(process);
        string user = nameof(user);
        string machine = nameof(machine);

        // Set the environment variable for each target.
        Console.WriteLine("Setting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, dft);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, process,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, user,

EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, machine,

EnvironmentVariableTarget.Machine);

        // Define an array of environment variables.
        string[] envVars = { defaultEnvVar,processEnvVar, userEnvVar,
machineEnvVar };

        // Try to get the environment variables from each target.
        // The default (no specified target).
        Console.WriteLine("Retrieving environment variables from the
```

```
default target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar) ?? "
(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The process block.
        Console.WriteLine("\nRetrieving environment variables from the
Process target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Process) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The user block.
        Console.WriteLine("\nRetrieving environment variables from the
User target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.User) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The machine block.
        Console.WriteLine("\nRetrieving environment variables from the
Machine target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Machine) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }

        // Delete the environment variable for each target.
        Console.WriteLine("\nDeleting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, null);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, null,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, null,

EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, null,

EnvironmentVariableTarget.Machine);
    }
}
// The example displays the following output if run on a Windows system:
```

```
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: machine
//
//      Deleting environment variables for each target...
//
// The example displays the following output if run on a Unix-based
system:
//
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
```

```
//        machineEnvVar: (none)
//
//      Deleting environment variables for each target...
```

Remarks

The names and values of the environment variables are stored as key/value pairs in the returned IDictionary object.

On Windows systems

On Windows systems, the `target` parameter specifies whether the source is the current process, the registry key for the current user, or the registry key for the local machine.

On non-Windows systems

On non-Windows systems, only a `target` value of EnvironmentVariableTarget.Process is supported. Per-process environment variables are inherited from the parent process (typically the shell) used to launch the `dotnet` process or are defined within the scope of the `dotnet` process itself. Once the dotnet process ends, these latter environment variables cease to exist.

Per-machine and per-user environment variables are not supported. A `target` value of EnvironmentVariableTarget.Machine or EnvironmentVariableTarget.User returns an empty array.

See also

- SetEnvironmentVariable(String, String, EnvironmentVariableTarget)
- EnvironmentVariableTarget

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.GetFolderPath Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the path to the system special folder that is identified by the specified enumeration.

Overloads

Expand table

GetFolderPath(Environment+SpecialFolder)	Gets the path to the specified system special folder.
GetFolderPath(Environment+SpecialFolder, Environment+SpecialFolderOption)	Gets the path to the specified system special folder using a specified option for accessing special folders.

GetFolderPath(Environment+SpecialFolder)

Gets the path to the specified system special folder.

C#

```
public static string GetFolderPath (Environment.SpecialFolder folder);
```

Parameters

`folder` Environment.SpecialFolder

One of the enumeration values that identifies a system special folder.

Returns

String

The path to the specified system special folder, if that folder physically exists on your computer; otherwise, an empty string ("").

A folder will not physically exist if the operating system did not create it, the existing folder was deleted, or the folder is a virtual directory, such as My Computer, which does not correspond to a physical path.

Exceptions

ArgumentException

`folder` is not a member of Environment.SpecialFolder.

PlatformNotSupportedException

The current platform is not supported.

Examples

The following example demonstrates how to use the GetFolderPath method to return and display the path associated with the `folder` parameter.

C#

```
// Sample for the Environment.GetFolderPath method
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    Console.WriteLine("GetFolderPath: {0}",

Environment.GetFolderPath(Environment.SpecialFolder.System));
    }
}
/*
This example produces the following results:

GetFolderPath: C:\WINNT\System32
*/
```

Remarks

This method retrieves the path to a system special folder, such as Program Files, Programs, System, or Startup, which can be used to access common information.

Special folders are set by default by the system, or explicitly by the user, when installing a version of Windows.

The `folder` parameter designates the special folder to retrieve and must be one of the values in the Environment.SpecialFolder enumeration; any other value throws an exception.

For more information about special folders, see Constant special item ID list (CSIDL).

See also

- GetFolderPath behavior on Unix

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetFolderPath(Environment+SpecialFolder, Environment+SpecialFolderOption)

Gets the path to the specified system special folder using a specified option for accessing special folders.

C#

```
public static string GetFolderPath (Environment.SpecialFolder folder,
Environment.SpecialFolderOption option);
```

Parameters

`folder` Environment.SpecialFolder

One of the enumeration values that identifies a system special folder.

`option` Environment.SpecialFolderOption

One of the enumeration values that specifies options to use for accessing a special folder.

Returns

String

The path to the specified system special folder, if that folder physically exists on your computer; otherwise, an empty string ("").

A folder will not physically exist if the operating system did not create it, the existing folder was deleted, or the folder is a virtual directory, such as My Computer, which does not correspond to a physical path.

Exceptions

ArgumentException

`folder` is not a member of Environment.SpecialFolder.

-or-

`options` is not a member of Environment.SpecialFolderOption.

PlatformNotSupportedException

The current platform is not supported.

Remarks

This method retrieves the path to a system special folder, such as Program Files, Programs, System, or Startup, which can be used to access common information. Special folders are set by default by the system, or explicitly by the user, when installing a version of Windows.

The `folder` parameter designates the special folder to retrieve and must be one of the values in the Environment.SpecialFolder enumeration; any other value throws an exception.

For more information about special folders, see Constant special item ID list (CSIDL).

See also

- GetFolderPath behavior on Unix

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.GetLogicalDrives Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an array of string containing the names of the logical drives on the current computer.

C#

```
public static string[] GetLogicalDrives ();
```

Returns

String[]

An array of strings where each element contains the name of a logical drive. For example, if the computer's hard drive is the first logical drive, the first element returned is "C:\".

Exceptions

IOException

An I/O error occurs.

SecurityException

The caller does not have the required permissions.

Examples

The following example shows how to display the logical drives of the current computer using the GetLogicalDrives method.

C#

```
// Sample for the Environment.GetLogicalDrives method
using System;
```

```
class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    String[] drives = Environment.GetLogicalDrives();
    Console.WriteLine("GetLogicalDrives: {0}", String.Join(", ", drives));
    }
}
/*
This example produces the following results:

GetLogicalDrives: A:\, C:\, D:\
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.SetEnvironmentVariable Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates, modifies, or deletes an environment variable.

Overloads

Expand table

SetEnvironmentVariable(String, String)	Creates, modifies, or deletes an environment variable stored in the current process.
SetEnvironmentVariable(String, String, EnvironmentVariableTarget)	Creates, modifies, or deletes an environment variable stored in the current process or in the Windows operating system registry key reserved for the current user or local machine.

SetEnvironmentVariable(String, String)

Creates, modifies, or deletes an environment variable stored in the current process.

C#

```
public static void SetEnvironmentVariable (string variable, string?
value);
```

Parameters

`variable` String

The name of an environment variable.

`value` String

A value to assign to `variable`.

Exceptions

ArgumentNullException

`variable` is `null`.

ArgumentException

`variable` contains a zero-length string, an initial hexadecimal zero character (0x00), or an equal sign ("=").

-or-

The length of `variable` or `value` is greater than or equal to 32,767 characters.

-or-

An error occurred during the execution of this operation.

SecurityException

The caller does not have the required permission to perform this operation.

Examples

The following example attempts to retrieve the value of an environment variable named `Test1` from the process environment block. If the variable doesn't exist, the example creates the variable and retrieves its value. The example displays the value of the variable. For .NET implementations running on Windows systems, it also calls the GetEnvironmentVariables(EnvironmentVariableTarget) method with each member of the EnvironmentVariableTarget enumeration to establish that the variable can be retrieved only from the current process environment block. (.NET implementations on Unix-based systems only support variables in the process environment block.) Finally, if the example created the variable, it deletes it.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string value;
      bool toDelete = false;
```

```
      // Check whether the environment variable exists.
      value = Environment.GetEnvironmentVariable("Test1");
      // If necessary, create it.
      if (value == null)
      {
         Environment.SetEnvironmentVariable("Test1", "Value1");
         toDelete = true;

         // Now retrieve it.
         value = Environment.GetEnvironmentVariable("Test1");
      }
      // Display the value.
      Console.WriteLine($"Test1: {value}\n");

      // Confirm that the value can only be retrieved from the process
      // environment block if running on a Windows system.
      if (Environment.OSVersion.Platform == PlatformID.Win32NT)
      {
         Console.WriteLine("Attempting to retrieve Test1 from:");
         foreach (EnvironmentVariableTarget enumValue in

Enum.GetValues(typeof(EnvironmentVariableTarget))) {
            value = Environment.GetEnvironmentVariable("Test1",
enumValue);
            Console.WriteLine($"   {enumValue}: {(value != null ? "found"
: "not found")}");
         }
         Console.WriteLine();
      }

      // If we've created it, now delete it.
      if (toDelete) {
         Environment.SetEnvironmentVariable("Test1", null);
         // Confirm the deletion.
         if (Environment.GetEnvironmentVariable("Test1") == null)
            Console.WriteLine("Test1 has been deleted.");
      }
   }
}
// The example displays the following output if run on a Windows system:
//      Test1: Value1
//
//      Attempting to retrieve Test1 from:
//         Process: found
//         User: not found
//         Machine: not found
//
//      Test1 has been deleted.
//
// The example displays the following output if run on a Unix-based
system:
//      Test1: Value1
//
//      Test1 has been deleted.
```

Remarks

Calling this method is equivalent to calling the SetEnvironmentVariable(String, String, EnvironmentVariableTarget) overload with a value of EnvironmentVariableTarget.Process for the `target` argument.

On non-Windows systems, calls to the SetEnvironmentVariable(String, String) method have no effect on any native libraries that are, or will be, loaded.

If the `value` argument is not empty (see the discussion of deleting an environment variable later in this section for the definition of an empty value) and the environment variable named by the `variable` parameter does not exist, the environment variable is created and assigned the contents of `value`. If it does exist, its value is modified. Because the environment variable is defined in the environment block of the current process only, it does not persist after the process has ended.

If `variable` contains a non-initial hexadecimal zero character, the characters before the zero character are considered the environment variable name and all subsequent characters are ignored.

If `value` contains a non-initial hexadecimal zero character, the characters before the zero character are assigned to the environment variable and all subsequent characters are ignored.

If `value` is empty and the environment variable named by `variable` exists, the environment variable is deleted. If `variable` does not exist, no error occurs even though the operation cannot be performed. `value` is considered empty under any of the following conditions:

- It is `null`.
- It is String.Empty.
- It consists of a single character whose value is U+0000.

See also

- GetEnvironmentVariable(String)
- GetEnvironmentVariables()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

SetEnvironmentVariable(String, String, EnvironmentVariableTarget)

Creates, modifies, or deletes an environment variable stored in the current process or in the Windows operating system registry key reserved for the current user or local machine.

C#

```
public static void SetEnvironmentVariable (string variable, string?
value, EnvironmentVariableTarget target);
```

Parameters

`variable` String

The name of an environment variable.

`value` String

A value to assign to `variable`.

`target` EnvironmentVariableTarget

One of the enumeration values that specifies the location of the environment variable.

Exceptions

ArgumentNullException

`variable` is `null`.

ArgumentException

`variable` contains a zero-length string, an initial hexadecimal zero character (0x00), or an equal sign ("=").

-or-

The length of `variable` is greater than or equal to 32,767 characters.

-or-

`target` is not a member of the EnvironmentVariableTarget enumeration.

-or-

`target` is Machine or User, and the length of `variable` is greater than or equal to 255.

-or-

`target` is Process and the length of `value` is greater than or equal to 32,767 characters.

-or-

An error occurred during the execution of this operation.

SecurityException
The caller does not have the required permission to perform this operation.

Examples

The following example creates environment variables for the EnvironmentVariableTarget.Process, EnvironmentVariableTarget.User, and Machine targets, checks whether the operating system registry contains the user and machine environment variables, then deletes the environment variables. Because .NET on Unix-based systems does not support per-user and per-machine environment variables, only SetEnvironmentVariable(String, String) and SetEnvironmentVariable(String, String, EnvironmentVariableTarget) with a value of EnvironmentVariableTarget.Process successfully store an environment variable to the process environment block.

C#

```
using System;
using System.Collections;
using Microsoft.Win32;
```

```
class Sample
{
    public static void Main()
    {
        // Environment variable names for default, process, user, and
machine targets.
        string defaultEnvVar = nameof(defaultEnvVar);
        string processEnvVar = nameof(processEnvVar);
        string userEnvVar = nameof(userEnvVar);
        string machineEnvVar = nameof(machineEnvVar);

        string dft = nameof(dft);
        string process = nameof(process);
        string user = nameof(user);
        string machine = nameof(machine);

        // Set the environment variable for each target.
        Console.WriteLine("Setting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, dft);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, process,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, user,

EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, machine,

EnvironmentVariableTarget.Machine);

        // Define an array of environment variables.
        string[] envVars = { defaultEnvVar,processEnvVar, userEnvVar,
machineEnvVar };

        // Try to get the environment variables from each target.
        // The default (no specified target).
        Console.WriteLine("Retrieving environment variables from the
default target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar) ?? "
(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The process block.
        Console.WriteLine("\nRetrieving environment variables from the
Process target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
```

```
EnvironmentVariableTarget.Process) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The user block.
        Console.WriteLine("\nRetrieving environment variables from the
User target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.User) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The machine block.
        Console.WriteLine("\nRetrieving environment variables from the
Machine target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Machine) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }

        // Delete the environment variable for each target.
        Console.WriteLine("\nDeleting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, null);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, null,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, null,

EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, null,

EnvironmentVariableTarget.Machine);
    }
}
// The example displays the following output if run on a Windows system:
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
```

```
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: machine
//
//      Deleting environment variables for each target...
//
// The example displays the following output if run on a Unix-based
system:
//
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Deleting environment variables for each target...
```

Remarks

The SetEnvironmentVariable(String, String, EnvironmentVariableTarget) method lets you define an environment variable that is available to the current process (the

Process value). Environment variables that are unique to the current process environment block persist only until the process ends.

In addition, on Windows systems only, the SetEnvironmentVariable(String, String, EnvironmentVariableTarget) method lets you define an environment variable that is available to all processes that run on a machine (the EnvironmentVariableTarget.Machine value) and to all processes run by a user (the EnvironmentVariableTarget.User value). Per-machine and per-user environment variables are copied into the environment block of the current process.

On non-Windows systems, calls to the SetEnvironmentVariable(String, String, EnvironmentVariableTarget) method with a value of EnvironmentVariableTarget.Machine or EnvironmentVariableTarget.User are ignored.

On non-Windows systems, calls to the SetEnvironmentVariable(String, String, EnvironmentVariableTarget) method with a value of EnvironmentVariableTarget.Process have no effect on any native libraries that are, or will be, loaded.

If the `value` argument is not empty (see the discussion of deleting an environment variable later in this section for the definition of an empty value) and the environment variable named by the `variable` argument does not exist, the environment variable is created and assigned the contents of `value`. If it does exist, its value is modified.

If `variable` contains a non-initial hexadecimal zero character, the characters before the zero character are considered the environment variable name and all subsequent characters are ignored.

If `value` contains a non-initial hexadecimal zero character, the characters before the zero character are assigned to the environment variable and all subsequent characters are ignored.

If `value` is empty and the environment variable named by `variable` exists, the environment variable is deleted. `value` is considered empty under any of the following conditions:

- It is `null`.
- It is String.Empty.
- It consists of a single character whose value is U+0000.

If `variable` does not exist, no error occurs although the operation cannot be performed. Be careful when `target` is Machine, because you can accidentally delete an environment variable that affects your entire local machine, not just the current process or user.

EnvironmentVariableTarget.Machine and EnvironmentVariableTarget.User on Windows systems

If `target` is EnvironmentVariableTarget.User, the environment variable is stored in the HKEY_CURRENT_USER\Environment key of the local computer's registry. It is also copied to instances of File Explorer that are running as the current user. The environment variable is then inherited by any new processes that the user launches from File Explorer.

Similarly, if `target` is EnvironmentVariableTarget.Machine, the environment variable is stored in the HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Session Manager\Environment key of the local computer's registry. It is also copied to all instances of File Explorer. The environment variable is then inherited by any new processes that are launched from File Explorer.

If `target` is User or Machine, other applications are notified of the set operation by a Windows `WM_SETTINGCHANGE` message.

If `target` is EnvironmentVariableTarget.User or EnvironmentVariableTarget.Machine, we recommend that the length of `value` be less than 2048 characters.

See also

- GetEnvironmentVariable(String, EnvironmentVariableTarget)
- GetEnvironmentVariables(EnvironmentVariableTarget)
- EnvironmentVariableTarget

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.SpecialFolder Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies enumerated constants used to retrieve directory paths to system special folders.

C#

```
public enum Environment.SpecialFolder
```

Inheritance Object → ValueType → Enum → Environment.SpecialFolder

Fields

Expand table

AdminTools	48	The file system directory that is used to store administrative tools for an individual user. The Microsoft Management Console (MMC) will save customized consoles to this directory, and it will roam with the user.
ApplicationData	26	The directory that serves as a common repository for application-specific data for the current roaming user. A roaming user works on more than one computer on a network. A roaming user's profile is kept on a server on the network and is loaded onto a system when the user logs on.
CDBurning	59	The file system directory that acts as a staging area for files waiting to be written to a CD.
CommonAdminTools	47	The file system directory that contains administrative tools for all users of the computer.
CommonApplicationData	35	The directory that serves as a common repository for application-specific data that is used by all users.
CommonDesktopDirectory	25	The file system directory that contains files and folders that appear on the desktop for all users.

CommonDocuments	46	The file system directory that contains documents that are common to all users.
CommonMusic	53	The file system directory that serves as a repository for music files common to all users.
CommonOemLinks	58	This value is recognized in Windows Vista for backward compatibility, but the special folder itself is no longer used.
CommonPictures	54	The file system directory that serves as a repository for image files common to all users.
CommonProgramFiles	43	The directory for components that are shared across applications. To get the x86 common program files directory in a non-x86 process, use the ProgramFilesX86 member.
CommonProgramFilesX86	44	The **Program Files** folder.
CommonPrograms	23	A folder for components that are shared across applications.
CommonStartMenu	22	The file system directory that contains the programs and folders that appear on the **Start** menu for all users.
CommonStartup	24	The file system directory that contains the programs that appear in the **Startup** folder for all users.
CommonTemplates	45	The file system directory that contains the templates that are available to all users.
CommonVideos	55	The file system directory that serves as a repository for video files common to all users.
Cookies	33	The directory that serves as a common repository for Internet cookies.
Desktop	0	The logical Desktop rather than the physical file system location.
DesktopDirectory	16	The directory used to physically store file objects on the desktop. Do not confuse this directory with the desktop folder itself, which is a virtual folder.
Favorites	6	The directory that serves as a common repository for the user's favorite items.
Fonts	20	A virtual folder that contains fonts.
History	34	The directory that serves as a common repository for Internet history items.

InternetCache	32	The directory that serves as a common repository for temporary Internet files.
LocalApplicationData	28	The directory that serves as a common repository for application-specific data that is used by the current, non-roaming user.
LocalizedResources	57	The file system directory that contains localized resource data.
MyComputer	17	The **My Computer** folder. When passed to the `Environment.GetFolderPath` method, the `MyComputer` enumeration member always yields the empty string ("") because no path is defined for the My Computer folder.
MyDocuments	5	The **My Documents** folder. This member is equivalent to Personal.
MyMusic	13	The **My Music** folder.
MyPictures	39	The **My Pictures** folder.
MyVideos	14	The file system directory that serves as a repository for videos that belong to a user.
NetworkShortcuts	19	A file system directory that contains the link objects that may exist in the **My Network Places** virtual folder.
Personal	5	The directory that serves as a common repository for documents. This member is equivalent to MyDocuments.
PrinterShortcuts	27	The file system directory that contains the link objects that can exist in the **Printers** virtual folder.
ProgramFiles	38	The program files directory. In a non-x86 process, passing ProgramFiles to the GetFolderPath(Environment+SpecialFolder) method returns the path for non-x86 programs. To get the x86 program files directory in a non-x86 process, use the ProgramFilesX86 member.
ProgramFilesX86	42	The x86 **Program Files** folder.
Programs	2	The directory that contains the user's program groups.
Recent	8	The directory that contains the user's most recently used documents.
Resources	56	The file system directory that contains resource data.
SendTo	9	The directory that contains the Send To menu items.

StartMenu	11	The directory that contains the Start menu items.
Startup	7	The directory that corresponds to the user's Startup program group. The system starts these programs whenever a user logs on or starts Windows.
System	37	The System directory.
SystemX86	41	The Windows **System** folder.
Templates	21	The directory that serves as a common repository for document templates.
UserProfile	40	The user's profile folder. Applications should not create files or folders at this level; they should put their data under the locations referred to by ApplicationData.
Windows	36	The Windows directory or SYSROOT. This corresponds to the %windir% or %SYSTEMROOT% environment variables.

Examples

The following example shows how to use Environment.SpecialFolder with the Environment.GetFolderPath method to get the System directory.

C#

```
// Sample for the Environment.GetFolderPath method
using System;

class Sample
{
    public static void Main()
    {
    Console.WriteLine();
    Console.WriteLine("GetFolderPath: {0}",

Environment.GetFolderPath(Environment.SpecialFolder.System));
    }
}
/*
This example produces the following results:

GetFolderPath: C:\WINNT\System32
*/
```

Remarks

The system special folders are folders such as **Program Files**, **Programs**, **System**, or **Startup**, which contain common information. Special folders are set by default by the system, or explicitly by the user, when installing a version of Windows.

The Environment.GetFolderPath method returns the locations associated with this enumeration. The locations of these folders can have different values on different operating systems, the user can change some of the locations, and the locations are localized.

For more information about special folders, see the KNOWNFOLDERID constants in the Windows documentation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Environment.SpecialFolderOption Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies options to use for getting the path to a special folder.

C#

```
public enum Environment.SpecialFolderOption
```

Inheritance Object → ValueType → Enum → Environment.SpecialFolderOption

Fields

Expand table

Create	32768	The path to the folder is created if it does not already exist.
DoNotVerify	16384	The path to the folder is returned without verifying whether the path exists. If the folder is located on a network, specifying this option can reduce lag time.
None	0	The path to the folder is verified. If the folder exists, the path is returned. If the folder does not exist, an empty string is returned. This is the default behavior.

Remarks

The System.Environment.SpecialFolderOption enumeration is used to define the precise behavior of the Environment.GetFolderPath method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EnvironmentVariableTarget Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the location where an environment variable is stored or retrieved in a set or get operation.

```
C#

public enum EnvironmentVariableTarget
```

Inheritance Object → ValueType → Enum → EnvironmentVariableTarget

Fields

Expand table

Machine	2	The environment variable is stored or retrieved from the `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\Environment` key in the Windows operating system registry. This value should be used on .NET implementations running on Windows systems only.
Process	0	The environment variable is stored or retrieved from the environment block associated with the current process.
User	1	The environment variable is stored or retrieved from the `HKEY_CURRENT_USER\Environment` key in the Windows operating system registry. This value should be used on .NET implementations running on Windows systems only.

Examples

The following example uses the EnvironmentVariableTarget enumeration in methods that create, retrieve, and delete environment variables. The output from the example shows that environmnent variables stored and retrieved without specifying a `EnvironmentVariableTarget` value are stored in the environment block associated with the current process (`EnvironmentVariableTarget.Process`). The example output from

Unix-based systems also shows that attempts to define an environment variable with a value other than `EnvironmentVariableTarget.Process` is ignored.

C#

```
using System;
using System.Collections;
using Microsoft.Win32;

class Sample
{
    public static void Main()
    {
        // Environment variable names for default, process, user, and
machine targets.
        string defaultEnvVar = nameof(defaultEnvVar);
        string processEnvVar = nameof(processEnvVar);
        string userEnvVar = nameof(userEnvVar);
        string machineEnvVar = nameof(machineEnvVar);

        string dft = nameof(dft);
        string process = nameof(process);
        string user = nameof(user);
        string machine = nameof(machine);

        // Set the environment variable for each target.
        Console.WriteLine("Setting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, dft);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, process,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, user,
                                           EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, machine,

EnvironmentVariableTarget.Machine);

        // Define an array of environment variables.
        string[] envVars = { defaultEnvVar,processEnvVar, userEnvVar,
machineEnvVar };

        // Try to get the environment variables from each target.
        // The default (no specified target).
        Console.WriteLine("Retrieving environment variables from the default
target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar) ?? "
(none)";
```

```
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The process block.
        Console.WriteLine("\nRetrieving environment variables from the
Process target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Process) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The user block.
        Console.WriteLine("\nRetrieving environment variables from the User
target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.User) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }
        // The machine block.
        Console.WriteLine("\nRetrieving environment variables from the
Machine target:");
        foreach (var envVar in envVars)
        {
          var value = Environment.GetEnvironmentVariable(envVar,
EnvironmentVariableTarget.Machine) ?? "(none)";
          Console.WriteLine($"   {envVar}: {value}");
        }

        // Delete the environment variable for each target.
        Console.WriteLine("\nDeleting environment variables for each
target...\n");
        // The default target (the current process).
        Environment.SetEnvironmentVariable(defaultEnvVar, null);
        // The current process.
        Environment.SetEnvironmentVariable(processEnvVar, null,

EnvironmentVariableTarget.Process);
        // The current user.
        Environment.SetEnvironmentVariable(userEnvVar, null,
                                           EnvironmentVariableTarget.User);
        // The local machine.
        Environment.SetEnvironmentVariable(machineEnvVar, null,

EnvironmentVariableTarget.Machine);
    }
}
// The example displays the following output if run on a Windows system:
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
```

```
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: user
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: machine
//
//      Deleting environment variables for each target...
//
// The example displays the following output if run on a Unix-based system:
//
//      Setting environment variables for each target...
//
//      Retrieving environment variables from the default target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Process target:
//        defaultEnvVar: dft
//        processEnvVar: process
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the User target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Retrieving environment variables from the Machine target:
//        defaultEnvVar: (none)
//        processEnvVar: (none)
//        userEnvVar: (none)
//        machineEnvVar: (none)
//
//      Deleting environment variables for each target...
```

Remarks

The EnvironmentVariableTarget enumeration is used by certain overloads of the Environment.SetEnvironmentVariable, Environment.GetEnvironmentVariable, and Environment.GetEnvironmentVariables methods to specify the location, or target, where the name and value of an environment variable is stored or retrieved.

The target can be one of three locations:

- The environment block associated with the current process (`EnvironmentVariableTarget.Process`).

  The user creates the environment variable in a set operation. When the process terminates, the operating system destroys the environment variable in that process.

- The Windows operating system registry key reserved for environment variables associated with the current user (`EnvironmentVariableTarget.User`).

  On Windows systems, when the user creates the environment variable in a set operation, the operating system stores the environment variable in the system registry, but not in the current process. If the user starts a new process, the operating system copies the environment variable from the registry to that process. When the process terminates, the operating system destroys the environment variable in that process. However, the environment variable in the registry persists until the user removes it programmatically or by means of an operating system tool.

  On Unix-based systems, an attempt to create an enviroment variable with `EnvironmentVariable.User` has no effect, and an attempt to retrieve an enviroment variable using `EnvironmentVariable.User` returns `null` (in C#) or `Nothing` (in Visual Basic).

- The registry key reserved for environment variables associated with all users on the local machine (`EnvironmentVariableTarget.Machine`).

  When a user creates the environment variable in a set operation, the operating system stores the environment variable in the system registry, but not in the current process. If any user on the local machine starts a new process, the operating system copies the environment variable from the registry to that process. When the process terminates, the operating system destroys the environment variable in that process. However, the environment variable in the

registry persists until a user removes it programmatically or by means of an operating system tool.

On Unix-based systems, an attempt to create an enviroment variable with `EnvironmentVariable.Machine` has no effect, and an attempt to retrieve an enviroment variable using `EnvironmentVariable.Machine` returns `null` (in C#) or `Nothing` (in Visual Basic).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EventArgs Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the base class for classes that contain event data, and provides a value to use for events that do not include event data.

C#

```
public class EventArgs
```

Inheritance Object → EventArgs

Derived Microsoft.VisualBasic.ApplicationServices.ApplyApplicationDefaultsEventArgs
Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs
Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs
Microsoft.Win32.PowerModeChangedEventArgs
Microsoft.Win32.SessionEndedEventArgs
More...

Examples

The following example shows a custom event data class named `ThresholdReachedEventArgs` that derives from the EventArgs class. An instance of the event data class is passed to the event handler for the `ThresholdReached` event.

C#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;
```

```
            Console.WriteLine("press 'a' key to increase total");
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }

        static void c_ThresholdReached(object sender,
ThresholdReachedEventArgs e)
        {
            Console.WriteLine("The threshold of {0} was reached at {1}.",
e.Threshold,  e.TimeReached);
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReachedEventArgs args = new
ThresholdReachedEventArgs();
                args.Threshold = threshold;
                args.TimeReached = DateTime.Now;
                OnThresholdReached(args);
            }
        }

        protected virtual void OnThresholdReached(ThresholdReachedEventArgs
e)
        {
            EventHandler<ThresholdReachedEventArgs> handler =
ThresholdReached;
            if (handler != null)
            {
                handler(this, e);
            }
        }

        public event EventHandler<ThresholdReachedEventArgs>
ThresholdReached;
    }
```

```
    public class ThresholdReachedEventArgs : EventArgs
    {
        public int Threshold { get; set; }
        public DateTime TimeReached { get; set; }
    }
}
```

Remarks

This class serves as the base class for all classes that represent event data. For example, the System.AssemblyLoadEventArgs class derives from EventArgs and is used to hold the data for assembly load events. To create a custom event data class, create a class that derives from the EventArgs class and provide the properties to store the necessary data. The name of your custom event data class should end with `EventArgs`.

To pass an object that does not contain any data, use the Empty field.

For more information about events, see the Handling and Raising Events article.

Constructors

Expand table

EventArgs()	Initializes a new instance of the EventArgs class.

Fields

Expand table

Empty	Provides a value to use with events that do not have event data.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)

GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Handling and Raising Events
- How to: Raise and Consume Events
- Events (Visual Basic)
- Events (C# Programming Guide)
- Events and routed events overview (Windows store apps)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EventArgs Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the EventArgs class.

C#

```
public EventArgs ();
```

Remarks

To pass an object that does not contain any data, use the Empty field.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Handling and Raising Events
- How to: Raise and Consume Events
- Events (Visual Basic)
- Events (C# Programming Guide)
- Events and routed events overview (Windows store apps)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EventArgs.Empty Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a value to use with events that do not have event data.

C#

```
public static readonly EventArgs Empty;
```

Field Value

EventArgs

Examples

The following example shows a simple counting application that raises an event when a threshold is equaled or exceeded. The Empty field is passed to the `OnThresholdReached` method.

C#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }
```

```
        static void c_ThresholdReached(object sender, EventArgs e)
        {
            Console.WriteLine("The threshold was reached.");
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                OnThresholdReached(EventArgs.Empty);
            }
        }

        protected virtual void OnThresholdReached(EventArgs e)
        {
            EventHandler handler = ThresholdReached;
            if (handler != null)
            {
                handler(this, e);
            }
        }

        public event EventHandler ThresholdReached;
    }
}
```

Remarks

Pass this value to event handlers that are associated with events that do not have data.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Handling and Raising Events
- How to: Raise and Consume Events
- Events (Visual Basic)
- Events (C# Programming Guide)
- Events (F#)
- Events and routed events overview (Windows store apps)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EventHandler Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that will handle an event that has no event data.

C#

```
public delegate void EventHandler(object? sender, EventArgs e);
```

Parameters

sender Object

The source of the event.

e EventArgs

An object that contains no event data.

Examples

The following example shows an event named `ThresholdReached` that is associated with an EventHandler delegate. The method assigned to the EventHandler delegate is called in the `OnThresholdReached` method.

C#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
```

```
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }

        static void c_ThresholdReached(object sender,
ThresholdReachedEventArgs e)
        {
            Console.WriteLine("The threshold of {0} was reached at {1}.",
e.Threshold,  e.TimeReached);
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReachedEventArgs args = new
ThresholdReachedEventArgs();
                args.Threshold = threshold;
                args.TimeReached = DateTime.Now;
                OnThresholdReached(args);
            }
        }

        protected virtual void OnThresholdReached(ThresholdReachedEventArgs
e)
        {
            EventHandler<ThresholdReachedEventArgs> handler =
ThresholdReached;
            if (handler != null)
            {
                handler(this, e);
            }
        }

        public event EventHandler<ThresholdReachedEventArgs>
ThresholdReached;
    }

    public class ThresholdReachedEventArgs : EventArgs
```

```
    {
        public int Threshold { get; set; }
        public DateTime TimeReached { get; set; }
    }
}
```

Remarks

The event model in the .NET Framework is based on having an event delegate that connects an event with its handler. To raise an event, two elements are needed:

- A delegate that identifies the method that provides the response to the event.
- Optionally, a class that holds the event data, if the event provides data.

The delegate is a type that defines a signature, that is, the return value type and parameter list types for a method. You can use the delegate type to declare a variable that can refer to any method with the same signature as the delegate.

The standard signature of an event handler delegate defines a method that does not return a value. This method's first parameter is of type Object and refers to the instance that raises the event. Its second parameter is derived from type EventArgs and holds the event data. If the event does not generate event data, the second parameter is simply the value of the EventArgs.Empty field. Otherwise, the second parameter is a type derived from EventArgs and supplies any fields or properties needed to hold the event data.

The EventHandler delegate is a predefined delegate that specifically represents an event handler method for an event that does not generate data. If your event does generate data, you must use the generic EventHandler<TEventArgs> delegate class.

To associate the event with the method that will handle the event, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate.

For more information about event handler delegates, see Handling and Raising Events.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- EventHandler<TEventArgs>
- EventArgs
- Delegate
- Handling and Raising Events
- How to: Raise and Consume Events
- Events (Visual Basic)
- Events (C# Programming Guide)
- Events (F#)
- Events and routed events overview (Windows store apps)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EventHandler<TEventArgs> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that will handle an event when the event provides data.

C#

```
public delegate void EventHandler<TEventArgs>(object? sender, TEventArgs e);
```

Type Parameters

TEventArgs

The type of the event data generated by the event.

Parameters

sender Object

The source of the event.

e TEventArgs

An object that contains the event data.

Examples

The following example shows an event named `ThresholdReached`. The event is associated with an EventHandler<TEventArgs> delegate.

C#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }

        static void c_ThresholdReached(object sender,
ThresholdReachedEventArgs e)
        {
            Console.WriteLine("The threshold of {0} was reached at {1}.",
e.Threshold,  e.TimeReached);
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReachedEventArgs args = new
ThresholdReachedEventArgs();
                args.Threshold = threshold;
                args.TimeReached = DateTime.Now;
                OnThresholdReached(args);
            }
        }

        protected virtual void OnThresholdReached(ThresholdReachedEventArgs
e)
        {
            EventHandler<ThresholdReachedEventArgs> handler =
ThresholdReached;
            if (handler != null)
            {
                handler(this, e);
            }
        }
```

```
        public event EventHandler<ThresholdReachedEventArgs>
ThresholdReached;
    }

    public class ThresholdReachedEventArgs : EventArgs
    {
        public int Threshold { get; set; }
        public DateTime TimeReached { get; set; }
    }
}
```

Remarks

The event model in the .NET Framework is based on having an event delegate that connects an event with its handler. To raise an event, two elements are needed:

- A delegate that refers to a method that provides the response to the event.
- Optionally, a class that holds the event data, if the event provides data.

The delegate is a type that defines a signature, that is, the return value type and parameter list types for a method. You can use the delegate type to declare a variable that can refer to any method with the same signature as the delegate.

The standard signature of an event handler delegate defines a method that does not return a value. This method's first parameter is of type Object and refers to the instance that raises the event. Its second parameter is derived from type EventArgs and holds the event data. If the event does not generate event data, the second parameter is simply the value of the EventArgs.Empty field. Otherwise, the second parameter is a type derived from EventArgs and supplies any fields or properties needed to hold the event data.

The EventHandler<TEventArgs> delegate is a predefined delegate that represents an event handler method for an event that generates data. The advantage of using EventHandler<TEventArgs> is that you do not need to code your own custom delegate if your event generates event data. You simply provide the type of the event data object as the generic parameter.

To associate the event with the method that will handle the event, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate.

For more information about event handler delegates, see Handling and Raising Events.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- EventHandler
- EventArgs
- Delegate
- Handling and Raising Events
- How to: Raise and Consume Events
- Events (Visual Basic)
- Events (C# Programming Guide)
- Events (F#)
- Events and routed events overview (Windows store apps)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents errors that occur during application execution.

C#

```
public class Exception : System.Runtime.Serialization.ISerializable
```

Inheritance Object → Exception

Derived Microsoft.CSharp.RuntimeBinder.RuntimeBinderException
Microsoft.CSharp.RuntimeBinder.RuntimeBinderInternalCompilerException
Microsoft.Extensions.CommandLineUtils.CommandParsingException
Microsoft.Extensions.Hosting.HostAbortedException
Microsoft.Extensions.Options.OptionsValidationException
More...

Implements ISerializable

Remarks

For more information about this API, see Supplemental API remarks for Exception.

Constructors

Expand table

Exception()	Initializes a new instance of the Exception class.
Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the Exception class with serialized data.

Exception(String)	Initializes a new instance of the Exception class with a specified error message.
Exception(String, Exception)	Initializes a new instance of the Exception class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
HelpLink	Gets or sets a link to the help file associated with this exception.
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
Inner Exception	Gets the Exception instance that caused the current exception.
Message	Gets a message that describes the current exception.
Source	Gets or sets the name of the application or the object that causes the error.
StackTrace	Gets a string representation of the immediate frames on the call stack.
TargetSite	Gets the method that throws the current exception.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType()	Gets the runtime type of the current instance.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception.

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Handling and Throwing Exceptions
- Packaging and Deploying Resources in Desktop Apps
- Assertions in Managed Code

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Exception Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Exception class.

Overloads

Expand table

Exception()	Initializes a new instance of the Exception class.
Exception(String)	Initializes a new instance of the Exception class with a specified error message.
Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the Exception class with serialized data.
Exception(String, Exception)	Initializes a new instance of the Exception class with a specified error message and a reference to the inner exception that is the cause of this exception.

Exception()

Initializes a new instance of the Exception class.

C#

```
public Exception ();
```

Examples

The following code example derives an `Exception` that uses a predefined message. The code demonstrates the use of the parameterless constructor for the derived

class and the base `Exception` class.

C#

```
// Example for the Exception( ) constructor.
using System;

namespace NDP_UE_CS
{
    // Derive an exception with a predefined message.
    class NotEvenException : Exception
    {
        public NotEvenException( ) :
            base( "The argument to a function requiring " +
                "even input is not divisible by 2." )
        { }
    }

    class NewExceptionDemo
    {
        public static void Main()
        {
            Console.WriteLine(
                "This example of the Exception( ) constructor " +
                "generates the following output." );
            Console.WriteLine(
                "\nHere, an exception is thrown using the \n" +
                "parameterless constructor of the base class.\n" );

            CalcHalf( 12 );
            CalcHalf( 15 );

            Console.WriteLine(
                "\nHere, an exception is thrown using the \n" +
                "parameterless constructor of a derived class.\n" );

            CalcHalf2( 24 );
            CalcHalf2( 27 );
        }

        // Half throws a base exception if the input is not even.
        static int Half( int input )
        {
            if( input % 2 != 0 )
                throw new Exception( );

            else return input / 2;
        }

        // Half2 throws a derived exception if the input is not even.
        static int Half2( int input )
        {
            if( input % 2 != 0 )
                throw new NotEvenException( );
```

```
            else return input / 2;
        }

        // CalcHalf calls Half and catches any thrown exceptions.
        static void CalcHalf(int input )
        {
            try
            {
                int halfInput = Half( input );
                Console.WriteLine(
                    "Half of {0} is {1}.", input, halfInput );
            }
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }

        // CalcHalf2 calls Half2 and catches any thrown exceptions.
        static void CalcHalf2(int input )
        {
            try
            {
                int halfInput = Half2( input );
                Console.WriteLine(
                    "Half of {0} is {1}.", input, halfInput );
            }
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }
    }
}

/*
This example of the Exception( ) constructor generates the following
output.

Here, an exception is thrown using the
parameterless constructor of the base class.

Half of 12 is 6.
System.Exception: Exception of type System.Exception was thrown.
   at NDP_UE_CS.NewExceptionDemo.Half(Int32 input)
   at NDP_UE_CS.NewExceptionDemo.CalcHalf(Int32 input)

Here, an exception is thrown using the
parameterless constructor of a derived class.

Half of 24 is 12.
NDP_UE_CS.NotEvenException: The argument to a function requiring even
input is
not divisible by 2.
```

```
    at NDP_UE_CS.NewExceptionDemo.Half2(Int32 input)
    at NDP_UE_CS.NewExceptionDemo.CalcHalf2(Int32 input)
*/
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error and takes into account the current system culture.

All the derived classes should provide this parameterless constructor. The following table shows the initial property values for an instance of Exception.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	A system-supplied localized description.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Exception(String)

Initializes a new instance of the Exception class with a specified error message.

C#

```
public Exception (string? message);
```

Parameters

`message` String

The message that describes the error.

Examples

The following code example derives an `Exception` for a specific condition. The code demonstrates the use of the constructor that takes a caller-specified message as a parameter, for both the derived class and the base `Exception` class.

C#

```
// Example for the Exception( string ) constructor.
using System;

namespace NDP_UE_CS2
{
    // Derive an exception with a specifiable message.
    class NotEvenException : Exception
    {
        const string notEvenMessage =
            "The argument to a function requiring " +
            "even input is not divisible by 2.";

        public NotEvenException( ) :
            base( notEvenMessage )
        { }

        public NotEvenException( string auxMessage ) :
            base( String.Format( "{0} - {1}",
                auxMessage, notEvenMessage ) )
        { }
    }

    class NewSExceptionDemo
    {
        public static void Main()
        {
            Console.WriteLine(
                "This example of the Exception( string )\n" +
                "constructor generates the following output." );
            Console.WriteLine(
                "\nHere, an exception is thrown using the \n" +
                "constructor of the base class.\n" );

            CalcHalf( 18 );
```

```
            CalcHalf( 21 );

            Console.WriteLine(
                "\nHere, an exception is thrown using the \n" +
                "constructor of a derived class.\n" );

            CalcHalf2( 30 );
            CalcHalf2( 33 );
        }

        // Half throws a base exception if the input is not even.
        static int Half( int input )
        {
            if( input % 2 != 0 )
                throw new Exception( String.Format(
                    "The argument {0} is not divisible by 2.",
                    input ) );

            else return input / 2;
        }

        // Half2 throws a derived exception if the input is not even.
        static int Half2( int input )
        {
            if( input % 2 != 0 )
                throw new NotEvenException(
                    String.Format( "Invalid argument: {0}", input ) );

            else return input / 2;
        }

        // CalcHalf calls Half and catches any thrown exceptions.
        static void CalcHalf(int input )
        {
            try
            {
                int halfInput = Half( input );
                Console.WriteLine(
                    "Half of {0} is {1}.", input, halfInput );
            }
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }

        // CalcHalf2 calls Half2 and catches any thrown exceptions.
        static void CalcHalf2(int input )
        {
            try
            {
                int halfInput = Half2( input );
                Console.WriteLine(
                    "Half of {0} is {1}.", input, halfInput );
            }
```

```
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }
    }
}

/*
This example of the Exception( string )
constructor generates the following output.

Here, an exception is thrown using the
constructor of the base class.

Half of 18 is 9.
System.Exception: The argument 21 is not divisible by 2.
   at NDP_UE_CS.NewSExceptionDemo.Half(Int32 input)
   at NDP_UE_CS.NewSExceptionDemo.CalcHalf(Int32 input)

Here, an exception is thrown using the
constructor of a derived class.

Half of 30 is 15.
NDP_UE_CS.NotEvenException: Invalid argument: 33 - The argument to a
function r
equiring even input is not divisible by 2.
   at NDP_UE_CS.NewSExceptionDemo.Half2(Int32 input)
   at NDP_UE_CS.NewSExceptionDemo.CalcHalf2(Int32 input)
*/
```

Remarks

This constructor initializes the Message property of the new instance by using the `message` parameter. If the `message` parameter is `null`, this is the same as calling the Exception constructor.

The following table shows the initial property values for an instance of Exception.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Exception(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the Exception class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected Exception (System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

SerializationException

The class name is `null` or HResult is zero (0).

Examples

The following code example defines a derived serializable `Exception` class. The code forces a divide-by-0 error and then creates an instance of the derived exception using the (SerializationInfo, StreamingContext) constructor. The code serializes the instance to a file, deserializes the file into a new exception, which it throws, and then catches and displays the exception's data.

C#

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;

// Define a serializable derived exception class.
[Serializable]
class SecondLevelException : Exception, ISerializable
{
    // This public constructor is used by class instantiators.
    public SecondLevelException(string message, Exception inner) :
        base(message, inner)
    {
        HelpLink = "http://MSDN.Microsoft.com";
        Source = "Exception_Class_Samples";
    }

    // This protected constructor is used for deserialization.
    protected SecondLevelException(SerializationInfo info,
        StreamingContext context) :
            base(info, context)
    { }

    // GetObjectData performs a custom serialization.
    public override void GetObjectData(SerializationInfo info,
        StreamingContext context)
    {
        // Change the case of two properties, and then use the
```

```
        // method of the base class.
        HelpLink = HelpLink.ToLower();
        Source = Source.ToUpperInvariant();

        base.GetObjectData(info, context);
    }
}

class SerializationDemo
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of the Exception constructor " +
            "and Exception.GetObjectData\nwith Serialization" +
            "Info and StreamingContext parameters " +
            "generates \nthe following output.\n");

        try
        {
            // This code forces a division by 0 and catches the
            // resulting exception.
            try
            {
                int zero = 0;
                int ecks = 1 / zero;
            }
            catch (Exception ex)
            {
                // Create a new exception to throw again.
                SecondLevelException newExcept =
                    new SecondLevelException(
                        "Forced a division by 0 and threw " +
                        "another exception.", ex);

                Console.WriteLine(
                    "Forced a division by 0, caught the " +
                    "resulting exception, \n" +
                    "and created a derived exception:\n");
                Console.WriteLine("HelpLink: {0}",
                    newExcept.HelpLink);
                Console.WriteLine("Source:   {0}",
                    newExcept.Source);

                // This FileStream is used for the serialization.
                FileStream stream =
                    new FileStream("NewException.dat",
                        FileMode.Create);

                try
                {
                    // Serialize the derived exception.
                    SoapFormatter formatter =
                        new SoapFormatter(null,
                            new StreamingContext(
```

```
                                StreamingContextStates.File));
                    formatter.Serialize(stream, newExcept);

                    // Rewind the stream and deserialize the
                    // exception.
                    stream.Position = 0;
                    SecondLevelException deserExcept =
                        (SecondLevelException)
                            formatter.Deserialize(stream);

                    Console.WriteLine(
                        "\nSerialized the exception, and then " +
                        "deserialized the resulting stream " +
                        "into a \nnew exception. " +
                        "The deserialization changed the case " +
                        "of certain properties:\n");

                    // Throw the deserialized exception again.
                    throw deserExcept;
                }
                catch (SerializationException se)
                {
                    Console.WriteLine("Failed to serialize: {0}",
                        se.ToString());
                }
                finally
                {
                    stream.Close();
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("HelpLink: {0}", ex.HelpLink);
            Console.WriteLine("Source:   {0}", ex.Source);

            Console.WriteLine();
            Console.WriteLine(ex.ToString());
        }
    }
}
/*
This example displays the following output.

Forced a division by 0, caught the resulting exception,
and created a derived exception:

HelpLink: http://MSDN.Microsoft.com
Source:   Exception_Class_Samples

Serialized the exception, and then deserialized the resulting stream into
a
new exception. The deserialization changed the case of certain
properties:
```

```
HelpLink: http://msdn.microsoft.com
Source:   EXCEPTION_CLASS_SAMPLES

NDP_UE_CS.SecondLevelException: Forced a division by 0 and threw another
except
ion. ---> System.DivideByZeroException: Attempted to divide by zero.
   at NDP_UE_CS.SerializationDemo.Main()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.SerializationDemo.Main()
*/
```

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- SerializationInfo
- StreamingContext
- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Exception(String, Exception)

Initializes a new instance of the Exception class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public Exception (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception, or a null reference (`Nothing` in Visual Basic) if no inner exception is specified.

Examples

The following code example derives an `Exception` for a specific condition. The code demonstrates the use of the constructor that takes a message and an inner exception as parameters, for both the derived class and the base `Exception` class.

C#

```
// Example for the Exception( string, Exception ) constructor.
using System;

namespace NDP_UE_CS
{
    // Derive an exception with a specifiable message and inner
exception.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage =
            "The log table has overflowed.";

        public LogTableOverflowException( ) :
            base( overflowMessage )
        { }

        public LogTableOverflowException( string auxMessage ) :
            base( String.Format( "{0} - {1}",
                overflowMessage, auxMessage ) )
        { }

        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
        { }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
```

```
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception
        // if the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception ex )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), ex );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of the Exception( string, Exception )" +
                "\nconstructor generates the following output." );
            Console.WriteLine(
                "\nExample of a derived exception " +
                "that references an inner exception:\n" );
            try
            {
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
                }
            }
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }
    }
}

/*
```

```
This example of the Exception( string, Exception )
constructor generates the following output.

Example of a derived exception that references an inner exception:

NDP_UE_CS.LogTableOverflowException: The log table has overflowed. -
Record "Lo
g record number 5" was not logged. ---> System.IndexOutOfRangeException:
Index
was outside the bounds of the array.
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   --- End of inner exception stack trace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()
*/
```

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of Exception.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.Data Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a collection of key/value pairs that provide additional user-defined information about the exception.

C#

```
public virtual System.Collections.IDictionary Data { get; }
```

Property Value

IDictionary

An object that implements the IDictionary interface and contains a collection of user-defined key/value pairs. The default is an empty collection.

Examples

The following example demonstrates how to add and retrieve information using the Data property.

C#

```
// This example demonstrates the Exception.Data property.
using System;
using System.Collections;

class Sample
{
   public static void Main()
   {
      Console.WriteLine("\nException with some extra information...");
      RunTest(false);
      Console.WriteLine("\nException with all extra information...");
      RunTest(true);
   }

   public static void RunTest(bool displayDetails)
```

```
    {
        try {
            NestedRoutine1(displayDetails);
        }
        catch (Exception e) {
            Console.WriteLine("An exception was thrown.");
            Console.WriteLine(e.Message);
            if (e.Data.Count > 0) {
                Console.WriteLine("  Extra details:");
                foreach (DictionaryEntry de in e.Data)
                    Console.WriteLine("    Key: {0,-20}      Value: {1}",
                                      "'" + de.Key.ToString() + "'", de.Value);
            }
        }
    }

    public static void NestedRoutine1(bool displayDetails)
    {
        try {
            NestedRoutine2(displayDetails);
        }
        catch (Exception e) {
            e.Data["ExtraInfo"] = "Information from NestedRoutine1.";
            e.Data.Add("MoreExtraInfo", "More information from
NestedRoutine1.");
            throw;
        }
    }

    public static void NestedRoutine2(bool displayDetails)
    {
        Exception e = new Exception("This statement is the original exception
message.");
        if (displayDetails) {
            string s = "Information from NestedRoutine2.";
            int i = -903;
            DateTime dt = DateTime.Now;
            e.Data.Add("stringInfo", s);
            e.Data["IntInfo"] = i;
            e.Data["DateTimeInfo"] = dt;
        }
        throw e;
    }
}
// The example displays the following output:
//    Exception with some extra information...
//    An exception was thrown.
//    This statement is the original exception message.
//      Extra details:
//        Key: 'ExtraInfo'               Value: Information from
NestedRoutine1.
//        Key: 'MoreExtraInfo'           Value: More information from
NestedRoutine1.
//
//    Exception with all extra information...
```

```
//    An exception was thrown.
//    This statement is the original exception message.
//      Extra details:
//        Key: 'stringInfo'              Value: Information from
NestedRoutine2.
//        Key: 'IntInfo'                 Value: -903
//        Key: 'DateTimeInfo'            Value: 7/29/2013 10:50:13 AM
//        Key: 'ExtraInfo'               Value: Information from
NestedRoutine1.
//        Key: 'MoreExtraInfo'           Value: More information from
NestedRoutine1.
```

Remarks

For more information about this API, see Supplemental API remarks for Exception.Data.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- DictionaryEntry

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.HelpLink Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets a link to the help file associated with this exception.

C#

```
public virtual string? HelpLink { get; set; }
```

Property Value

String

The Uniform Resource Name (URN) or Uniform Resource Locator (URL).

Examples

The following code example throws an `Exception` that sets the `HelpLink` property in its constructor and then catches the exception and displays `HelpLink`.

C#

```
// Example for the Exception.HelpLink, Exception.Source,
// Exception.StackTrace, and Exception.TargetSite properties.
using System;

namespace NDP_UE_CS
{
    // Derive an exception; the constructor sets the HelpLink and
    // Source properties.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage = "The log table has overflowed.";

        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
        {
```

```
            this.HelpLink = "https://docs.microsoft.com";
            this.Source = "Exception_Class_Samples";
        }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception if
        // the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception e )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), e );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of \n   Exception.Message, \n" +
                "   Exception.HelpLink, \n   Exception.Source, \n" +
                "   Exception.StackTrace, and \n   Exception." +
                "TargetSite \ngenerates the following output." );

            try
            {
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
                }
```

```
            }
            catch( Exception ex )
            {
                Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
                Console.WriteLine(
                    "\nHelpLink ---\n{0}", ex.HelpLink );
                Console.WriteLine( "\nSource ---\n{0}", ex.Source );
                Console.WriteLine(
                    "\nStackTrace ---\n{0}", ex.StackTrace );
                Console.WriteLine(
                    "\nTargetSite ---\n{0}", ex.TargetSite );
            }
        }
    }
}

/*
This example of
   Exception.Message,
   Exception.HelpLink,
   Exception.Source,
   Exception.StackTrace, and
   Exception.TargetSite
generates the following output.

Message ---
The log table has overflowed. - Record "Log record number 5" was not logged.

HelpLink ---
https://docs.microsoft.com

Source ---
Exception_Class_Samples

StackTrace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()

TargetSite ---
Int32 AddRecord(System.String)
*/
```

Remarks

The return value, which represents a help file, is a URN or URL. For example, the `HelpLink` value could be:

"file:///C:/Applications/Bazzal/help.html#ErrorNum42"

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.HResult Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

C#

```
public int HResult { get; set; }
```

Property Value

Int32

The HRESULT value.

Examples

The following code example defines a derived `Exception` class that sets the `HResult` property to a custom value in its constructor.

C#

```
// Example for the Exception.HResult property.
using System;

namespace NDP_UE_CS
{
    // Create the derived exception class.
    class SecondLevelException : Exception
    {
        const int SecondLevelHResult = unchecked( (int)0x81234567 );

        // Set HResult for this exception, and include it in
        // the exception message.
        public SecondLevelException( string message, Exception inner ) :
            base( string.Format( "(HRESULT:0x{1:X8}) {0}",
                message, SecondLevelHResult ), inner )
        {
            HResult = SecondLevelHResult;
```

```
        }
    }

    class HResultDemo
    {
        public static void Main()
        {
            // This function forces a division by 0 and throws
            // a second exception.
            try
            {
                try
                {
                    int  zero = 0;
                    int  ecks = 1 / zero;
                }
                catch( Exception ex )
                {
                    throw new SecondLevelException(
                        "Forced a division by 0 and threw " +
                        "a second exception.", ex );
                }
            }
            catch( Exception ex )
            {
                Console.WriteLine( ex.ToString( ) );
            }
        }
    }
}

/*
This example of Exception.HResult generates the following output.

NDP_UE_CS.SecondLevelException: (HRESULT:0x81234567) Forced a division by 0
and
 threw a second exception. ---> System.DivideByZeroException: Attempted to
divi
de by zero.
   at NDP_UE_CS.HResultDemo.Main()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.HResultDemo.Main()
*/
```

Remarks

HRESULT is a 32-bit value, divided into three different fields: a severity code, a facility code, and an error code. The severity code indicates whether the return value represents information, warning, or error. The facility code identifies the area of the system responsible for the error. The error code is a unique number that is assigned to represent the exception. Each exception is mapped to a distinct HRESULT. When

managed code throws an exception, the runtime passes the HRESULT to the COM client. When unmanaged code returns an error, the HRESULT is converted to an exception, which is then thrown by the runtime. For information about HRESULT values and their corresponding .NET Framework exceptions, see How to: Map HRESULTs and Exceptions. See Common HRESULT Values in the Windows documentation for a list of the values you are most likely to encounter.

Starting with the .NET Framework 4.5, the HResult property's setter is protected, whereas its getter is public. In previous versions of the .NET Framework, both getter and setter are protected.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- How to: Map HRESULTs and Exceptions
- Common HRESULT Values

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.InnerException Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Exception instance that caused the current exception.

```
C#

public Exception? InnerException { get; }
```

Property Value

Exception

An object that describes the error that caused the current exception. The InnerException property returns the same value as was passed into the Exception(String, Exception) constructor, or `null` if the inner exception value was not supplied to the constructor. This property is read-only.

Examples

The following example demonstrates throwing and catching an exception that references an inner exception.

```
C#

using System;

public class AppException : Exception
{
   public AppException(String message) : base (message)
   {}

   public AppException(String message, Exception inner) :
base(message,inner) {}
}

public class Example
{
```

```
    public static void Main()
    {
       Example ex = new Example();

       try {
          ex.CatchInner();
       }
       catch(AppException e) {
          Console.WriteLine ("In catch block of Main method.");
          Console.WriteLine("Caught: {0}", e.Message);
          if (e.InnerException != null)
             Console.WriteLine("Inner exception: {0}", e.InnerException);
       }
    }

    public void ThrowInner ()
    {
       throw new AppException("Exception in ThrowInner method.");
    }

    public void CatchInner()
    {
       try {
          this.ThrowInner();
       }
       catch (AppException e) {
          throw new AppException("Error in CatchInner caused by calling the
ThrowInner method.", e);
       }
    }
}
// The example displays the following output:
//    In catch block of Main method.
//    Caught: Error in CatchInner caused by calling the ThrowInner method.
//    Inner exception: AppException: Exception in ThrowInner method.
//       at Example.ThrowInner()
//       at Example.CatchInner()
```

Remarks

When an exception `X` is thrown as a direct result of a previous exception `Y`, the InnerException property of `X` should contain a reference to `Y`.

Use the InnerException property to obtain the set of exceptions that led to the current exception.

You can create a new exception that catches an earlier exception. The code that handles the second exception can make use of the additional information from the earlier exception to handle the error more appropriately.

Suppose that there is a function that reads a file and formats the data from that file. In this example, as the code tries to read the file, an IOException is thrown. The function catches the IOException and throws a FileNotFoundException. The IOException could be saved in the InnerException property of the FileNotFoundException, enabling the code that catches the FileNotFoundException to examine the cause of the initial error.

The InnerException property, which holds a reference to the inner exception, is set upon initialization of the exception object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a message that describes the current exception.

C#

```
public virtual string Message { get; }
```

Property Value

String

The error message that explains the reason for the exception, or an empty string ("").

Examples

The following code example throws and then catches an Exception exception and displays the exception's text message using the Message property.

C#

```
// Example for the Exception.HelpLink, Exception.Source,
// Exception.StackTrace, and Exception.TargetSite properties.
using System;

namespace NDP_UE_CS
{
    // Derive an exception; the constructor sets the HelpLink and
    // Source properties.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage = "The log table has overflowed.";

        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
        {
```

```
            this.HelpLink = "https://docs.microsoft.com";
            this.Source = "Exception_Class_Samples";
        }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception if
        // the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception e )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), e );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of \n   Exception.Message, \n" +
                "   Exception.HelpLink, \n   Exception.Source, \n" +
                "   Exception.StackTrace, and \n   Exception." +
                "TargetSite \ngenerates the following output." );

            try
            {
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
                }
```

```
            }
            catch( Exception ex )
            {
                Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
                Console.WriteLine(
                    "\nHelpLink ---\n{0}", ex.HelpLink );
                Console.WriteLine( "\nSource ---\n{0}", ex.Source );
                Console.WriteLine(
                    "\nStackTrace ---\n{0}", ex.StackTrace );
                Console.WriteLine(
                    "\nTargetSite ---\n{0}", ex.TargetSite );
            }
        }
    }
}

/*
This example of
   Exception.Message,
   Exception.HelpLink,
   Exception.Source,
   Exception.StackTrace, and
   Exception.TargetSite
generates the following output.

Message ---
The log table has overflowed. - Record "Log record number 5" was not logged.

HelpLink ---
https://docs.microsoft.com

Source ---
Exception_Class_Samples

StackTrace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()

TargetSite ---
Int32 AddRecord(System.String)
*/
```

Remarks

Error messages target the developer who is handling the exception. The text of the Message property should completely describe the error and, when possible, should also explain how to correct the error. Top-level exception handlers may display the message to end-users, so you should ensure that it is grammatically correct and that each sentence of the message ends with a period. Do not use question marks or exclamation

points. If your application uses localized exception messages, you should ensure that they are accurately translated.

> **Important**
>
> Do not disclose sensitive information in exception messages without checking for the appropriate permissions.

The value of the Message property is included in the information returned by ToString. The Message property is set only when creating an Exception. If no message was supplied to the constructor for the current instance, the system supplies a default message that is formatted using the current system culture.

Notes to Inheritors

The Message property is overridden in classes that require control over message content or format. Application code typically accesses this property when it needs to display information about an exception that has been caught.

The error message should be localized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Exception.Source Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the name of the application or the object that causes the error.

C#

```
public virtual string? Source { get; set; }
```

Property Value

String

The name of the application or the object that causes the error.

Exceptions

ArgumentException

The object must be a runtime System.Reflection object.

Examples

The following example throws an `Exception` that sets the `Source` property in its constructor and then catches the exception and displays `Source`.

C#

```
// Example for the Exception.HelpLink, Exception.Source,
// Exception.StackTrace, and Exception.TargetSite properties.
using System;

namespace NDP_UE_CS
{
    // Derive an exception; the constructor sets the HelpLink and
    // Source properties.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage = "The log table has overflowed.";
```

```
        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
        {
            this.HelpLink = "https://docs.microsoft.com";
            this.Source = "Exception_Class_Samples";
        }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception if
        // the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception e )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), e );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of \n   Exception.Message, \n" +
                "   Exception.HelpLink, \n   Exception.Source, \n" +
                "   Exception.StackTrace, and \n   Exception." +
                "TargetSite \ngenerates the following output." );

            try
            {
```

```
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
                }
            }
            catch( Exception ex )
            {
                Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
                Console.WriteLine(
                    "\nHelpLink ---\n{0}", ex.HelpLink );
                Console.WriteLine( "\nSource ---\n{0}", ex.Source );
                Console.WriteLine(
                    "\nStackTrace ---\n{0}", ex.StackTrace );
                Console.WriteLine(
                    "\nTargetSite ---\n{0}", ex.TargetSite );
            }
        }
    }
}

/*
This example of
   Exception.Message,
   Exception.HelpLink,
   Exception.Source,
   Exception.StackTrace, and
   Exception.TargetSite
generates the following output.

Message ---
The log table has overflowed. - Record "Log record number 5" was not logged.

HelpLink ---
https://docs.microsoft.com

Source ---
Exception_Class_Samples

StackTrace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()

TargetSite ---
Int32 AddRecord(System.String)
*/
```

Remarks

If the Source property is not set explicitly, the runtime automatically sets it to the name of the assembly in which the exception originated.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.StackTrace Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a string representation of the immediate frames on the call stack.

C#

```
public virtual string? StackTrace { get; }
```

Property Value

String

A string that describes the immediate frames of the call stack. If no stack trace is available (such as prior to stack unwinding from a `throw` statement), the value is `null`.

Examples

The following code example throws an `Exception` and then catches it and displays a stack trace using the `StackTrace` property.

C#

```
// Example for the Exception.HelpLink, Exception.Source,
// Exception.StackTrace, and Exception.TargetSite properties.
using System;

namespace NDP_UE_CS
{
    // Derive an exception; the constructor sets the HelpLink and
    // Source properties.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage = "The log table has overflowed.";

        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
```

```
        {
            this.HelpLink = "https://docs.microsoft.com";
            this.Source = "Exception_Class_Samples";
        }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception if
        // the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception e )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), e );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of \n   Exception.Message, \n" +
                "   Exception.HelpLink, \n   Exception.Source, \n" +
                "   Exception.StackTrace, and \n   Exception." +
                "TargetSite \ngenerates the following output." );

            try
            {
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
```

```
                }
            }
            catch( Exception ex )
            {
                Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
                Console.WriteLine(
                    "\nHelpLink ---\n{0}", ex.HelpLink );
                Console.WriteLine( "\nSource ---\n{0}", ex.Source );
                Console.WriteLine(
                    "\nStackTrace ---\n{0}", ex.StackTrace );
                Console.WriteLine(
                    "\nTargetSite ---\n{0}", ex.TargetSite );
            }
        }
    }
}

/*
This example of
   Exception.Message,
   Exception.HelpLink,
   Exception.Source,
   Exception.StackTrace, and
   Exception.TargetSite
generates the following output.

Message ---
The log table has overflowed. - Record "Log record number 5" was not logged.

HelpLink ---
https://docs.microsoft.com

Source ---
Exception_Class_Samples

StackTrace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()

TargetSite ---
Int32 AddRecord(System.String)
*/
```

Remarks

The execution stack keeps track of all the methods that are in execution at a given instant. A trace of the method calls is called a stack trace. The stack trace listing provides a way to follow the call stack to the line number in the method where the exception occurs.

The StackTrace property returns the frames of the call stack that originate at the location where the exception was thrown. You can obtain information about additional frames in the call stack by creating a new instance of the System.Diagnostics.StackTrace class and using its StackTrace.ToString method.

The common language runtime (CLR) updates the stack trace whenever an exception is thrown in application code (by using the `throw` keyword). If the exception was rethrown in a method that is different than the method where it was originally thrown, the stack trace contains both the location in the method where the exception was originally thrown, and the location in the method where the exception was rethrown. If the exception is thrown, and later rethrown, in the same method, the stack trace only contains the location where the exception was rethrown and does not include the location where the exception was originally thrown.

The StackTrace property may not report as many method calls as expected because of code transformations, such as inlining, that occur during optimization.

Notes to Inheritors

The `StackTrace` property is overridden in classes that require control over the stack trace content or format.

By default, the stack trace is captured immediately before an exception object is thrown. Use StackTrace to get stack trace information when no exception is being thrown.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- StackTrace

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Exception.TargetSite Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the method that throws the current exception.

C#

```
public System.Reflection.MethodBase? TargetSite { get; }
```

Property Value

MethodBase

The MethodBase that threw the current exception.

Examples

The following code example throws an `Exception` and then catches it and displays the originating method using the `TargetSite` property.

C#

```
// Example for the Exception.HelpLink, Exception.Source,
// Exception.StackTrace, and Exception.TargetSite properties.
using System;

namespace NDP_UE_CS
{
    // Derive an exception; the constructor sets the HelpLink and
    // Source properties.
    class LogTableOverflowException : Exception
    {
        const string overflowMessage = "The log table has overflowed.";

        public LogTableOverflowException(
            string auxMessage, Exception inner ) :
                base( String.Format( "{0} - {1}",
                    overflowMessage, auxMessage ), inner )
        {
```

```
            this.HelpLink = "https://docs.microsoft.com";
            this.Source = "Exception_Class_Samples";
        }
    }

    class LogTable
    {
        public LogTable( int numElements )
        {
            logArea = new string[ numElements ];
            elemInUse = 0;
        }

        protected string[ ] logArea;
        protected int       elemInUse;

        // The AddRecord method throws a derived exception if
        // the array bounds exception is caught.
        public    int       AddRecord( string newRecord )
        {
            try
            {
                logArea[ elemInUse ] = newRecord;
                return elemInUse++;
            }
            catch( Exception e )
            {
                throw new LogTableOverflowException(
                    String.Format( "Record \"{0}\" was not logged.",
                        newRecord ), e );
            }
        }
    }

    class OverflowDemo
    {
        // Create a log table and force an overflow.
        public static void Main()
        {
            LogTable log = new LogTable( 4 );

            Console.WriteLine(
                "This example of \n   Exception.Message, \n" +
                "   Exception.HelpLink, \n   Exception.Source, \n" +
                "   Exception.StackTrace, and \n   Exception." +
                "TargetSite \ngenerates the following output." );

            try
            {
                for( int count = 1; ; count++ )
                {
                    log.AddRecord(
                        String.Format(
                            "Log record number {0}", count ) );
                }
```

```
            }
            catch( Exception ex )
            {
                Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
                Console.WriteLine(
                    "\nHelpLink ---\n{0}", ex.HelpLink );
                Console.WriteLine( "\nSource ---\n{0}", ex.Source );
                Console.WriteLine(
                    "\nStackTrace ---\n{0}", ex.StackTrace );
                Console.WriteLine(
                    "\nTargetSite ---\n{0}", ex.TargetSite );
            }
        }
    }
}

/*
This example of
   Exception.Message,
   Exception.HelpLink,
   Exception.Source,
   Exception.StackTrace, and
   Exception.TargetSite
generates the following output.

Message ---
The log table has overflowed. - Record "Log record number 5" was not logged.

HelpLink ---
https://docs.microsoft.com

Source ---
Exception_Class_Samples

StackTrace ---
   at NDP_UE_CS.LogTable.AddRecord(String newRecord)
   at NDP_UE_CS.OverflowDemo.Main()

TargetSite ---
Int32 AddRecord(System.String)
*/
```

Remarks

If the method that throws this exception is not available and the stack trace is not a null reference (`Nothing` in Visual Basic), TargetSite obtains the method from the stack trace. If the stack trace is a null reference, TargetSite also returns a null reference.

> ⓘ **Note**

The **TargetSite** property may not accurately report the name of the method in which an exception was thrown if the exception handler handles an exception across application domain boundaries.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.GetBaseException Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.

C#

```
public virtual Exception GetBaseException ();
```

Returns

Exception

The first exception thrown in a chain of exceptions. If the InnerException property of the current exception is a null reference (`Nothing` in Visual Basic), this property returns the current exception.

Examples

The following code example defines two derived `Exception` classes. It forces an exception and then throws it again with each of the derived classes. The code shows the use of the `GetBaseException` method to retrieve the original exception.

C#

```
// Example for the Exception.GetBaseException method.
using System;

namespace NDP_UE_CS
{
    // Define two derived exceptions to demonstrate nested exceptions.
    class SecondLevelException : Exception
    {
        public SecondLevelException( string message, Exception inner )
            : base( message, inner )
        { }
    }
```

```
class ThirdLevelException : Exception
{
    public ThirdLevelException( string message, Exception inner )
        : base( message, inner )
    { }
}

class NestedExceptions
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of Exception.GetBaseException " +
            "generates the following output." );
        Console.WriteLine(
            "\nThe program forces a division by 0, then " +
            "throws the exception \ntwice more, " +
            "using a different derived exception each time.\n" );

        try
        {
            // This function calls another that forces a
            // division by 0.
            Rethrow( );
        }
        catch( Exception ex )
        {
            Exception current;

            Console.WriteLine(
                "Unwind the nested exceptions " +
                "using the InnerException property:\n" );

            // This code unwinds the nested exceptions using the
            // InnerException property.
            current = ex;
            while( current != null )
            {
                Console.WriteLine( current.ToString( ) );
                Console.WriteLine( );
                current = current.InnerException;
            }

            // Display the innermost exception.
            Console.WriteLine(
                "Display the base exception " +
                "using the GetBaseException method:\n" );
            Console.WriteLine(
                ex.GetBaseException( ).ToString( ) );
        }
    }

    // This function catches the exception from the called
    // function DivideBy0( ) and throws another in response.
    static void Rethrow()
```

```
        {
            try
            {
                DivideBy0( );
            }
            catch( Exception ex )
            {
                throw new ThirdLevelException(
                    "Caught the second exception and " +
                    "threw a third in response.", ex );
            }
        }

        // This function forces a division by 0 and throws a second
        // exception.
        static void DivideBy0( )
        {
            try
            {
                int  zero = 0;
                int  ecks = 1 / zero;
            }
            catch( Exception ex )
            {
                throw new SecondLevelException(
                    "Forced a division by 0 and threw " +
                    "a second exception.", ex );
            }
        }
    }
}

/*
This example of Exception.GetBaseException generates the following output.

The program forces a division by 0, then throws the exception
twice more, using a different derived exception each time.

Unwind the nested exceptions using the InnerException property:

NDP_UE_CS.ThirdLevelException: Caught the second exception and threw a third
in
 response. ---> NDP_UE_CS.SecondLevelException: Forced a division by 0 and
thre
w a second exception. ---> System.DivideByZeroException: Attempted to divide
by
 zero.
   at NDP_UE_CS.NestedExceptions.DivideBy0()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.NestedExceptions.DivideBy0()
   at NDP_UE_CS.NestedExceptions.Rethrow()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.NestedExceptions.Rethrow()
   at NDP_UE_CS.NestedExceptions.Main()
```

```
NDP_UE_CS.SecondLevelException: Forced a division by 0 and threw a second
excep
tion. ---> System.DivideByZeroException: Attempted to divide by zero.
   at NDP_UE_CS.NestedExceptions.DivideBy0()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.NestedExceptions.DivideBy0()
   at NDP_UE_CS.NestedExceptions.Rethrow()

System.DivideByZeroException: Attempted to divide by zero.
   at NDP_UE_CS.NestedExceptions.DivideBy0()

Display the base exception using the GetBaseException method:

System.DivideByZeroException: Attempted to divide by zero.
   at NDP_UE_CS.NestedExceptions.DivideBy0()
*/
```

Remarks

A chain of exceptions consists of a set of exceptions such that each exception in the chain was thrown as a direct result of the exception referenced in its `InnerException` property. For a given chain, there can be exactly one exception that is the root cause of all other exceptions in the chain. This exception is called the base exception and its `InnerException` property always contains a null reference.

For all exceptions in a chain of exceptions, the `GetBaseException` method must return the same object (the base exception).

Use the `GetBaseException` method when you want to find the root cause of an exception but do not need information about exceptions that may have occurred between the current exception and the first exception.

Notes to Inheritors

The `GetBaseException` method is overridden in classes that require control over the exception content or format.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

When overridden in a derived class, sets the SerializationInfo with information about the exception.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public virtual void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` parameter is a null reference (`Nothing` in Visual Basic).

Examples

The following code example defines a derived serializable `Exception` class that implements `GetObjectData`, which makes minor changes to two properties and then calls the base class to perform the serialization. The example forces a divide-by-0 error and then creates an instance of the derived exception. The code serializes the instance to a file, deserializes the file into a new exception, which it throws, and then catches and displays the exception's data.

C#

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;

// Define a serializable derived exception class.
[Serializable]
class SecondLevelException : Exception, ISerializable
{
    // This public constructor is used by class instantiators.
    public SecondLevelException(string message, Exception inner) :
        base(message, inner)
    {
        HelpLink = "http://MSDN.Microsoft.com";
        Source = "Exception_Class_Samples";
    }

    // This protected constructor is used for deserialization.
    protected SecondLevelException(SerializationInfo info,
        StreamingContext context) :
            base(info, context)
    { }

    // GetObjectData performs a custom serialization.
    public override void GetObjectData(SerializationInfo info,
        StreamingContext context)
```

```
    {
        // Change the case of two properties, and then use the
        // method of the base class.
        HelpLink = HelpLink.ToLower();
        Source = Source.ToUpperInvariant();

        base.GetObjectData(info, context);
    }
}

class SerializationDemo
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of the Exception constructor " +
            "and Exception.GetObjectData\nwith Serialization" +
            "Info and StreamingContext parameters " +
            "generates \nthe following output.\n");

        try
        {
            // This code forces a division by 0 and catches the
            // resulting exception.
            try
            {
                int zero = 0;
                int ecks = 1 / zero;
            }
            catch (Exception ex)
            {
                // Create a new exception to throw again.
                SecondLevelException newExcept =
                    new SecondLevelException(
                        "Forced a division by 0 and threw " +
                        "another exception.", ex);

                Console.WriteLine(
                    "Forced a division by 0, caught the " +
                    "resulting exception, \n" +
                    "and created a derived exception:\n");
                Console.WriteLine("HelpLink: {0}",
                    newExcept.HelpLink);
                Console.WriteLine("Source:   {0}",
                    newExcept.Source);

                // This FileStream is used for the serialization.
                FileStream stream =
                    new FileStream("NewException.dat",
                        FileMode.Create);

                try
                {
                    // Serialize the derived exception.
                    SoapFormatter formatter =
```

```
                        new SoapFormatter(null,
                            new StreamingContext(
                                StreamingContextStates.File));
                    formatter.Serialize(stream, newExcept);

                    // Rewind the stream and deserialize the
                    // exception.
                    stream.Position = 0;
                    SecondLevelException deserExcept =
                        (SecondLevelException)
                            formatter.Deserialize(stream);

                    Console.WriteLine(
                        "\nSerialized the exception, and then " +
                        "deserialized the resulting stream " +
                        "into a \nnew exception. " +
                        "The deserialization changed the case " +
                        "of certain properties:\n");

                    // Throw the deserialized exception again.
                    throw deserExcept;
                }
                catch (SerializationException se)
                {
                    Console.WriteLine("Failed to serialize: {0}",
                        se.ToString());
                }
                finally
                {
                    stream.Close();
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("HelpLink: {0}", ex.HelpLink);
            Console.WriteLine("Source:   {0}", ex.Source);

            Console.WriteLine();
            Console.WriteLine(ex.ToString());
        }
    }
}
/*
This example displays the following output.

Forced a division by 0, caught the resulting exception,
and created a derived exception:

HelpLink: http://MSDN.Microsoft.com
Source:   Exception_Class_Samples

Serialized the exception, and then deserialized the resulting stream into a
new exception. The deserialization changed the case of certain properties:
```

```
HelpLink: http://msdn.microsoft.com
Source:   EXCEPTION_CLASS_SAMPLES

NDP_UE_CS.SecondLevelException: Forced a division by 0 and threw another
except
ion. ---> System.DivideByZeroException: Attempted to divide by zero.
   at NDP_UE_CS.SerializationDemo.Main()
   --- End of inner exception stack trace ---
   at NDP_UE_CS.SerializationDemo.Main()
*/
```

Remarks

`GetObjectData` sets a SerializationInfo with all the exception object data targeted for serialization. During deserialization, the exception is reconstituted from the `SerializationInfo` transmitted over the stream.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.GetType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the runtime type of the current instance.

C#

```
public Type GetType ();
```

Returns

Type

A Type object that represents the exact runtime type of the current instance.

Remarks

The GetType method exists to support the .NET Framework infrastructure, and internally invokes the fundamental method, Object.GetType.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Type
- Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Exception.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates and returns a string representation of the current exception.

C#

```
public override string ToString ();
```

Returns

String

A string representation of the current exception.

Examples

The following example causes an exception and displays the result of calling ToString on that exception. Note that the Exception.ToString method is called implicitly when the Exception class instance appears in the argument list of the Console.WriteLine method.

C#

```
using System;

public class TestClass
{}

public class Example
{
   public static void Main()
   {
      var test = new TestClass();
      Object[] objectsToCompare = { test, test.ToString(), 123,
                                    123.ToString(), "some text",
                                    "Some Text" };
      string s = "some text";
      foreach (var objectToCompare in objectsToCompare) {
         try {
```

```
            int i = s.CompareTo(objectToCompare);
            Console.WriteLine("Comparing '{0}' with '{1}': {2}",
                              s, objectToCompare, i);
         }
         catch (ArgumentException e) {
            Console.WriteLine("Bad argument: {0} (type {1})",
                              objectToCompare,
                              objectToCompare.GetType().Name);
            Console.WriteLine("Exception information: {0}", e);
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//     Bad argument: TestClass (type TestClass)
//     Exception information: System.ArgumentException: Object must be of
type String.
//        at System.String.CompareTo(Object value)
//        at Example.Main()
//
//     Comparing 'some text' with 'TestClass': -1
//
//     Bad argument: 123 (type Int32)
//     Exception information: System.ArgumentException: Object must be of
type String.
//        at System.String.CompareTo(Object value)
//        at Example.Main()
//
//     Comparing 'some text' with '123': 1
//
//     Comparing 'some text' with 'some text': 0
//
//     Comparing 'some text' with 'Some Text': -1
```

Remarks

ToString returns a representation of the current exception that is intended to be understood by humans. Where the exception contains culture-sensitive data, the string representation returned by `ToString` is required to take into account the current system culture. Although there are no exact requirements for the format of the returned string, it should attempt to reflect the value of the object as perceived by the user.

The default implementation of ToString obtains the name of the class that threw the current exception, the message, the result of calling ToString on the inner exception, and the result of calling Environment.StackTrace. If any of these members is `null`, its value is not included in the returned string.

If there is no error message or if it is an empty string (""), then no error message is returned. The name of the inner exception and the stack trace are returned only if they are not `null`.

This method overrides Object.ToString.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Exception.SerializeObjectState Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> **Caution**
>
> BinaryFormatter serialization is obsolete and should not be used. See **https://aka.ms/binaryformatter** for more information.

Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception.

C#

```
[System.Obsolete("BinaryFormatter serialization is obsolete and should not be used. See https://aka.ms/binaryformatter for more information.", DiagnosticId="SYSLIB0011", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
protected event EventHandler<System.Runtime.Serialization.SafeSerializationEventArgs>? SerializeObjectState;
```

Event Type

EventHandler<SafeSerializationEventArgs>

Attributes ObsoleteAttribute

Remarks

The exception state object implements the ISafeSerializationData interface.

When the SerializeObjectState event is subscribed to, the exception is deserialized and created as an empty exception. The exception's constructor is not run, and the exception state is also deserialized. The CompleteDeserialization callback method of the exception

state object is then notified so that it can push deserialized data into the empty exception.

The SerializeObjectState event enables transparent exception types to serialize and deserialize exception data. Transparent code can execute commands within the bounds of the permission set it is operating within, but cannot execute, call, derive from, or contain critical code.

If the SerializeObjectState event is not subscribed to, deserialization occurs as usual using the Exception constructor.

Typically, a handler for the SerializeObjectState event is added in the exception's constructor to provide for its serialization. But because the constructor is not executed when the SerializeObjectState event handler executes, serializing a deserialized exception can throw a SerializationException exception when you try to deserialize the exception. To avoid this, you should also add the handler for the SerializeObjectState event in the ISafeSerializationData.CompleteDeserialization method. See the Examples section for an illustration.

Notes to Inheritors

If this event is subscribed to and used, all derived types that follow in the inheritance hierarchy must implement the same serialization mechanism.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ExecutionEngineException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> ExecutionEngineException previously indicated an unspecified fatal error in the runtime. The runtime no longer raises this exception so this type is obsolete.

The exception that is thrown when there is an internal error in the execution engine of the common language runtime. This class cannot be inherited.

C#

```
[System.Obsolete("ExecutionEngineException previously indicated an
unspecified fatal error in the runtime. The runtime no longer raises this
exception so this type is obsolete.")]
public sealed class ExecutionEngineException : SystemException
```

Inheritance Object → Exception → SystemException → ExecutionEngineException

Attributes ObsoleteAttribute

Remarks

There is no non-obsolete alternative to ExecutionEngineException. If further execution of your application cannot be sustained, use the FailFast method.

Constructors

Expand table

ExecutionEngineException()	Initializes a new instance of the ExecutionEngineException class.

ExecutionEngineException(String)	Initializes a new instance of the ExecutionEngineException class with a specified error message.
ExecutionEngineException(String, Exception)	Initializes a new instance of the ExecutionEngineException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
InnerException	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)

GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5 *(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

See also

- FailFast(String, Exception)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ExecutionEngineException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ExecutionEngineException class.

Overloads

Expand table

ExecutionEngineException()	Initializes a new instance of the ExecutionEngineException class.
ExecutionEngineException(String)	Initializes a new instance of the ExecutionEngineException class with a specified error message.
ExecutionEngineException(String, Exception)	Initializes a new instance of the ExecutionEngineException class with a specified error message and a reference to the inner exception that is the cause of this exception.

ExecutionEngineException()

Initializes a new instance of the ExecutionEngineException class.

C#

```
public ExecutionEngineException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Internal error occurred." This message takes into account the current system culture.

The following table shows the initial property values for an instance of ExecutionEngineException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecutionEngineException(String)

Initializes a new instance of the ExecutionEngineException class with a specified error message.

C#

```
public ExecutionEngineException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of ExecutionEngineException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ExecutionEngineException(String, Exception)

Initializes a new instance of the ExecutionEngineException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public ExecutionEngineException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is

raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of ExecutionEngineException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

FieldAccessException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an invalid attempt to access a private or protected field inside a class.

C#

```
public class FieldAccessException : MemberAccessException
```

Inheritance Object → Exception → SystemException → MemberAccessException → FieldAccessException

Remarks

This exception is typically thrown when the access level of a field in a class library is changed, and one or more assemblies referencing the library have not been recompiled.

FieldAccessException uses the HRESULT COR_E_FIELDACCESS, which has the value 0x80131507.

For a list of initial property values for an instance of FieldAccessException, see the FieldAccessException constructors.

> **Note**
>
> This exception is not included in the **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **MemberAccessException** instead.

Constructors

Expand table

FieldAccessException()	Initializes a new instance of the FieldAccessException class.
FieldAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the FieldAccessException class with serialized data.
FieldAccessException(String)	Initializes a new instance of the FieldAccessException class with a specified error message.
FieldAccessException(String, Exception)	Initializes a new instance of the FieldAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FieldAccessException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the FieldAccessException class.

Overloads

Expand table

FieldAccessException()	Initializes a new instance of the FieldAccessException class.
FieldAccessException(String)	Initializes a new instance of the FieldAccessException class with a specified error message.
FieldAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the FieldAccessException class with serialized data.
FieldAccessException(String, Exception)	Initializes a new instance of the FieldAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

FieldAccessException()

Initializes a new instance of the FieldAccessException class.

C#

```
public FieldAccessException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to access a field that is not accessible by the caller." This message takes the current system culture into account.

The following table shows the initial property values for an instance of FieldAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FieldAccessException(String)

Initializes a new instance of the FieldAccessException class with a specified error message.

C#

```
public FieldAccessException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

This constructor initializes the Message property of the new instance by using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor must ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of FieldAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FieldAccessException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the FieldAccessException class with serialized data.

```C#
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
protected FieldAccessException (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object that was transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

FieldAccessException(String, Exception)

Initializes a new instance of the FieldAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public FieldAccessException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or returns `null` if it does not supply the inner exception value to the constructor. The caller of this constructor must ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of FieldAccessException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FileStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the File scheme.

C#

```
public class FileStyleUriParser : UriParser
```

Inheritance Object → UriParser → FileStyleUriParser

Constructors

Expand table

FileStyleUriParser()	Creates a customizable parser based on the File scheme.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)

IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FileStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a customizable parser based on the File scheme.

C#

```
public FileStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FlagsAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that an enumeration can be treated as a bit field; that is, a set of flags.

C#

```
[System.AttributeUsage(System.AttributeTargets.Enum, Inherited=false)]
public class FlagsAttribute : Attribute
```

Inheritance Object → Attribute → FlagsAttribute

Attributes AttributeUsageAttribute

Remarks

For more information about this API, see Supplemental API remarks for FlagsAttribute.

Constructors

Expand table

FlagsAttribute()	Initializes a new instance of the FlagsAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

FlagsAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the FlagsAttribute class.

C#

```
public FlagsAttribute ();
```

Examples

The following example defines a `PhoneService` enumeration that represents forms of communication provided by a telephone company. It initializes three variables representing the service provided to three different households, and then indicates which households have no service, which households have only cell phone service, and which households have both cell phone and land line service. Finally, it implicitly calls the Enum.ToString(String) method to display the types of service provided to each household.

C#

```
using System;

[Flags]
public enum PhoneService
{
   None = 0,
   LandLine = 1,
   Cell = 2,
   Fax = 4,
   Internet = 8,
   Other = 16
}

public class Example
{
   public static void Main()
   {
      // Define three variables representing the types of phone service
```

```
      // in three households.
      var household1 = PhoneService.LandLine | PhoneService.Cell |
                       PhoneService.Internet;
      var household2 = PhoneService.None;
      var household3 = PhoneService.Cell | PhoneService.Internet;

      // Store the variables in an array for ease of access.
      PhoneService[] households = { household1, household2, household3 };

      // Which households have no service?
      for (int ctr = 0; ctr < households.Length; ctr++)
         Console.WriteLine("Household {0} has phone service: {1}",
                           ctr + 1,
                           households[ctr] == PhoneService.None ?
                               "No" : "Yes");
      Console.WriteLine();

      // Which households have cell phone service?
      for (int ctr = 0; ctr < households.Length; ctr++)
         Console.WriteLine("Household {0} has cell phone service: {1}",
                           ctr + 1,
                           (households[ctr] & PhoneService.Cell) ==
PhoneService.Cell ?
                              "Yes" : "No");
      Console.WriteLine();

      // Which households have cell phones and land lines?
      var cellAndLand = PhoneService.Cell | PhoneService.LandLine;
      for (int ctr = 0; ctr < households.Length; ctr++)
         Console.WriteLine("Household {0} has cell and land line service:
{1}",
                           ctr + 1,
                           (households[ctr] & cellAndLand) == cellAndLand ?
                              "Yes" : "No");
      Console.WriteLine();

      // List all types of service of each household?//
      for (int ctr = 0; ctr < households.Length; ctr++)
         Console.WriteLine("Household {0} has: {1:G}",
                           ctr + 1, households[ctr]);
      Console.WriteLine();
   }
}
// The example displays the following output:
//    Household 1 has phone service: Yes
//    Household 2 has phone service: No
//    Household 3 has phone service: Yes
//
//    Household 1 has cell phone service: Yes
//    Household 2 has cell phone service: No
//    Household 3 has cell phone service: Yes
//
//    Household 1 has cell and land line service: Yes
//    Household 2 has cell and land line service: No
//    Household 3 has cell and land line service: No
```

```
//
//    Household 1 has: LandLine, Cell, Internet
//    Household 2 has: None
//    Household 3 has: Cell, Internet
```

F#

```fsharp
open System

[<Flags>]
type PhoneService =
    | None = 0
    | LandLine = 1
    | Cell = 2
    | Fax = 4
    | Internet = 8
    | Other = 16

// Define three variables representing the types of phone service
// in three households.
let household1 =
    PhoneService.LandLine ||| PhoneService.Cell ||| PhoneService.Internet

let household2 =
    PhoneService.None

let household3 =
    PhoneService.Cell ||| PhoneService.Internet

// Store the variables in a list for ease of access.
let households =
    [ household1; household2; household3 ]

// Which households have no service?
for i = 0 to households.Length - 1 do
    printfn $"""Household {i + 1} has phone service: {if households[i] =
PhoneService.None then "No" else "Yes"}"""
printfn ""

// Which households have cell phone service?
for i = 0 to households.Length - 1 do
    printfn $"""Household {i + 1} has cell phone service: {if households[i]
&&& PhoneService.Cell = PhoneService.Cell then "Yes" else "No"}"""
printfn ""

// Which households have cell phones and land lines?
let cellAndLand =
    PhoneService.Cell ||| PhoneService.LandLine

for i = 0 to households.Length - 1 do
    printfn $"""Household {i + 1} has cell and land line service: {if
households[i] &&& cellAndLand = cellAndLand then "Yes" else "No"}"""
printfn ""
```

```
// List all types of service of each household?//
for i = 0 to households.Length - 1 do
    printfn $"Household {i + 1} has: {households[i]:G}"

// The example displays the following output:
//    Household 1 has phone service: Yes
//    Household 2 has phone service: No
//    Household 3 has phone service: Yes
//
//    Household 1 has cell phone service: Yes
//    Household 2 has cell phone service: No
//    Household 3 has cell phone service: Yes
//
//    Household 1 has cell and land line service: Yes
//    Household 2 has cell and land line service: No
//    Household 3 has cell and land line service: No
//
//    Household 1 has: LandLine, Cell, Internet
//    Household 2 has: None
//    Household 3 has: Cell, Internet
```

VB

```
<Flags()>
Public Enum PhoneService As Integer
   None = 0
   LandLine = 1
   Cell = 2
   Fax = 4
   Internet = 8
   Other = 16
End Enum

Module Example
   Public Sub Main()
      ' Define three variables representing the types of phone service
      ' in three households.
      Dim household1 As PhoneService = PhoneService.LandLine Or
                                       PhoneService.Cell Or
                                       PhoneService.Internet
      Dim household2 As PhoneService = PhoneService.None
      Dim household3 As PhoneService = PhoneService.Cell Or
                                       PhoneService.Internet

      ' Store the variables in an array for ease of access.
      Dim households() As PhoneService = { household1, household2,
                                           household3 }

      ' Which households have no service?
      For ctr As Integer = 0 To households.Length - 1
         Console.WriteLine("Household {0} has phone service: {1}",
                           ctr + 1,
```

```
                            If(households(ctr) = PhoneService.None,
                               "No", "Yes"))
      Next
      Console.WriteLine()

      ' Which households have cell phone service?
      For ctr As Integer = 0 To households.Length - 1
         Console.WriteLine("Household {0} has cell phone service: {1}",
                            ctr + 1,
                            If((households(ctr) And PhoneService.Cell) =
PhoneService.Cell,
                               "Yes", "No"))
      Next
      Console.WriteLine()

      ' Which households have cell phones and land lines?
      Dim cellAndLand As PhoneService = PhoneService.Cell Or
PhoneService.LandLine
      For ctr As Integer = 0 To households.Length - 1
         Console.WriteLine("Household {0} has cell and land line service:
{1}",
                            ctr + 1,
                            If((households(ctr) And cellAndLand) =
cellAndLand,
                               "Yes", "No"))
      Next
      Console.WriteLine()

      ' List all types of service of each household?'
      For ctr As Integer = 0 To households.Length - 1
         Console.WriteLine("Household {0} has: {1:G}",
                            ctr + 1, households(ctr))
      Next
      Console.WriteLine()
   End Sub
End Module
' The example displays the following output:
'    Household 1 has phone service: Yes
'    Household 2 has phone service: No
'    Household 3 has phone service: Yes
'
'    Household 1 has cell phone service: Yes
'    Household 2 has cell phone service: No
'    Household 3 has cell phone service: Yes
'
'    Household 1 has cell and land line service: Yes
'    Household 2 has cell and land line service: No
'    Household 3 has cell and land line service: No
'
'    Household 1 has: LandLine, Cell, Internet
'    Household 2 has: None
'    Household 3 has: Cell, Internet
```

The following example illustrates the use of the `FlagsAttribute` attribute and shows the effect on the ToString method of using `FlagsAttribute` on an Enum declaration.

C#

```
using System;

class Example
{
   // Define an Enum without FlagsAttribute.
   enum SingleHue : short
   {
      None = 0,
      Black = 1,
      Red = 2,
      Green = 4,
      Blue = 8
   };

   // Define an Enum with FlagsAttribute.
   [Flags]
   enum MultiHue : short
   {
      None = 0,
      Black = 1,
      Red = 2,
      Green = 4,
      Blue = 8
   };

   static void Main( )
   {
      // Display all possible combinations of values.
      Console.WriteLine(
           "All possible combinations of values without FlagsAttribute:");
      for(int val = 0; val <= 16; val++ )
         Console.WriteLine( "{0,3} - {1:G}", val, (SingleHue)val);

      // Display all combinations of values, and invalid values.
      Console.WriteLine(
           "\nAll possible combinations of values with FlagsAttribute:");
      for( int val = 0; val <= 16; val++ )
         Console.WriteLine( "{0,3} - {1:G}", val, (MultiHue)val);
   }
}
// The example displays the following output:
//       All possible combinations of values without FlagsAttribute:
//         0 - None
//         1 - Black
//         2 - Red
//         3 - 3
//         4 - Green
//         5 - 5
```

```
//          6 - 6
//          7 - 7
//          8 - Blue
//          9 - 9
//         10 - 10
//         11 - 11
//         12 - 12
//         13 - 13
//         14 - 14
//         15 - 15
//         16 - 16
//
//       All possible combinations of values with FlagsAttribute:
//          0 - None
//          1 - Black
//          2 - Red
//          3 - Black, Red
//          4 - Green
//          5 - Black, Green
//          6 - Red, Green
//          7 - Black, Red, Green
//          8 - Blue
//          9 - Black, Blue
//         10 - Red, Blue
//         11 - Black, Red, Blue
//         12 - Green, Blue
//         13 - Black, Green, Blue
//         14 - Red, Green, Blue
//         15 - Black, Red, Green, Blue
//         16 - 16
```

F#

```fsharp
open System

// Define an Enum without FlagsAttribute.
type SingleHue =
    | None = 0
    | Black = 1
    | Red = 2
    | Green = 4
    | Blue = 8

// Define an Enum with FlagsAttribute.
[<Flags>]
type MultiHue =
    | None = 0
    | Black = 1
    | Red = 2
    | Green = 4
    | Blue = 8

// Display all possible combinations of values.
```

```
printfn "All possible combinations of values without FlagsAttribute:"
for i = 0 to 16 do
    printfn $"{i,3} - {enum<SingleHue> i:G}"

// Display all combinations of values, and invalid values.
printfn "\nAll possible combinations of values with FlagsAttribute:"
for i = 0 to 16 do
    printfn $"{i,3} - {enum<MultiHue> i:G}"

// The example displays the following output:
//       All possible combinations of values without FlagsAttribute:
//         0 - None
//         1 - Black
//         2 - Red
//         3 - 3
//         4 - Green
//         5 - 5
//         6 - 6
//         7 - 7
//         8 - Blue
//         9 - 9
//        10 - 10
//        11 - 11
//        12 - 12
//        13 - 13
//        14 - 14
//        15 - 15
//        16 - 16
//
//       All possible combinations of values with FlagsAttribute:
//         0 - None
//         1 - Black
//         2 - Red
//         3 - Black, Red
//         4 - Green
//         5 - Black, Green
//         6 - Red, Green
//         7 - Black, Red, Green
//         8 - Blue
//         9 - Black, Blue
//        10 - Red, Blue
//        11 - Black, Red, Blue
//        12 - Green, Blue
//        13 - Black, Green, Blue
//        14 - Red, Green, Blue
//        15 - Black, Red, Green, Blue
//        16 - 16
```

VB

```
Module Example
   ' Define an Enum without FlagsAttribute.
   Enum SingleHue As Short
```

```
        None = 0
        Black = 1
        Red = 2
        Green = 4
        Blue = 8
    End Enum

    ' Define an Enum with FlagsAttribute.
    <Flags()>
    Enum MultiHue As Short
        None = 0
        Black = 1
        Red = 2
        Green = 4
        Blue = 8
    End Enum

    Sub Main()
        ' Display all possible combinations of values.
        Console.WriteLine(
             "All possible combinations of values without FlagsAttribute:")
        For val As Integer = 0 To 16
           Console.WriteLine("{0,3} - {1:G}", val, CType(val, SingleHue))
      Next
      Console.WriteLine()

      ' Display all combinations of values, and invalid values.
      Console.WriteLine(
           "All possible combinations of values with FlagsAttribute:")
      For val As Integer = 0 To 16
         Console.WriteLine( "{0,3} - {1:G}", val, CType(val, MultiHue))
      Next
    End Sub
End Module
' The example displays the following output:
'       All possible combinations of values without FlagsAttribute:
'         0 - None
'         1 - Black
'         2 - Red
'         3 - 3
'         4 - Green
'         5 - 5
'         6 - 6
'         7 - 7
'         8 - Blue
'         9 - 9
'        10 - 10
'        11 - 11
'        12 - 12
'        13 - 13
'        14 - 14
'        15 - 15
'        16 - 16
'
'       All possible combinations of values with FlagsAttribute:
```

```
'          0 - None
'          1 - Black
'          2 - Red
'          3 - Black, Red
'          4 - Green
'          5 - Black, Green
'          6 - Red, Green
'          7 - Black, Red, Green
'          8 - Blue
'          9 - Black, Blue
'         10 - Red, Blue
'         11 - Black, Red, Blue
'         12 - Green, Blue
'         13 - Black, Green, Blue
'         14 - Red, Green, Blue
'         15 - Black, Red, Green, Blue
'         16 - 16
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormatException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the format of an argument is invalid, or when a composite format string is not well formed.

```
C#
public class FormatException : SystemException
```

Inheritance Object → Exception → SystemException → FormatException

Derived System.IO.FileFormatException
System.Net.CookieException
System.Reflection.CustomAttributeFormatException
System.UriFormatException

Remarks

A FormatException exception can be thrown for one of the following reasons:

- In a call to a method that converts a string to some other data type, the string doesn't conform to the required pattern. This typically occurs when calling some methods of the Convert class and the `Parse` and `ParseExact` methods of some types.

  In most cases, particularly if the string that you're converting is input by a user or read from a file, you should use a `try/catch` (`try/with` in F#) block and handle the FormatException exception if the conversion is unsuccessful. You can also replace the call to the conversion method with a call to a `TryParse` or `TryParseExact` method, if one exists. However, a FormatException exception that is thrown when you're trying to parse a predefined or hard-coded string indicates a program error. In this case, you should correct the error rather than handle the exception.

The conversion of a string to the following types in the System namespace can throw a FormatException exception:

- Boolean. The Boolean.Parse(String) and Convert.ToBoolean(String) methods require the string to be converted to be "True", "true", "False", or "false". Any other value throws a FormatException exception.

- DateTime and DateTimeOffset. All date and time data is interpreted based on the formatting conventions of a particular culture: either the current culture (or, in some cases, the current application domain culture), the invariant culture, or a specified culture. When you call the DateTime.ParseExact(String, String, IFormatProvider, DateTimeStyles) and DateTimeOffset.ParseExact(String, String[], IFormatProvider, DateTimeStyles) methods, date and time data must also conform *exactly* to a pattern specified by one or more standard format strings or custom format strings that are provided as arguments in the method call. If it doesn't conform to an expected culture-specific pattern, a FormatException exception is thrown. This means that date and time data saved in a culture-specific format on one system might not parse successfully on another system.

  For more information about parsing dates and times, see Parsing Date and Time Strings and the documentation for the method that threw the exception.

- **GUIDs.** The string representation of a GUID must consist of 32 hexadecimal digits (0-F), and must be in one of the five formats output by the Guid.ToString method. For more information, see the Guid.Parse method.

- **Numeric types, including all signed integers, unsigned integers, and floating-point types.** The string to be parsed must consist of the Latin digits 0-9. A positive or negative sign, decimal separator, group separators, and currency symbol may also be permitted. Trying to parse a string that contains any other character always throws a FormatException exception.

  All numeric strings are interpreted based on the formatting conventions of a particular culture: either the current culture, the invariant culture, or a specified culture. As a result, a numeric string that is parsed by using the conventions of one culture might fail when using the conventions of another.

  For more information about parsing numeric strings, see Parsing Numeric Strings and the documentation for the specific method that threw the exception.

- **Time intervals.** The string to be parsed must be either in fixed culture-insensitive format or in a culture-sensitive format defined by the current culture,

the invariant culture, or a specified culture. If the string isn't in an appropriate format, or if, at the minimum, the days, hours, and minutes components of the time interval aren't present, the parsing method throws a FormatException exception. For more information, see the documentation for the TimeSpan parsing method that threw the exception.

- A type implements the IFormattable interface, which supports format strings that define how an object is converted to its string representation, and an invalid format string is used. This is most common in a formatting operation. In the following example, the "Q" standard format string is used in a composite format string to format a number. However, "Q" is not a valid standard format string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      decimal price = 169.32m;
      Console.WriteLine("The cost is {0:Q2}.", price);
   }
}
// The example displays the following output:
//    Unhandled Exception: System.FormatException: Format specifier was
invalid.
//       at System.Number.FormatDecimal(Decimal value, String format,
NumberFormatInfo info)
//       at System.Decimal.ToString(String format, IFormatProvider
provider)
//       at System.Text.StringBuilder.AppendFormat(IFormatProvider
provider, String format, Object[] args)
//       at System.IO.TextWriter.WriteLine(String format, Object arg0)
//       at System.IO.TextWriter.SyncTextWriter.WriteLine(String
format, Object arg0)
//       at Example.Main()
```

This exception results from a coding error. To correct the error, either remove the format string or substitute a valid one. The following example corrects the error by replacing the invalid format string with the "C" (currency) format string.

C#

```
using System;

public class Example
{
   public static void Main()
```

```
    {
        decimal price = 169.32m;
        Console.WriteLine("The cost is {0:C2}.", price);
    }
}
// The example displays the following output:
//    The cost is $169.32.
```

A FormatException exception can also be thrown by parsing methods, such as DateTime.ParseExact and Guid.ParseExact, that require the string to be parsed to conform exactly to the pattern specified by a format string. In the following example, the string representation of a GUID is expected to conform to the pattern specified by the "G" standard format string. However, the Guid structure's implementation of IFormattable does not support the "G" format string.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string guidString = "ba748d5c-ae5f-4cca-84e5-1ac5291c38cb";
        Console.WriteLine(Guid.ParseExact(guidString, "G"));
    }
}
// The example displays the following output:
//    Unhandled Exception: System.FormatException:
//       Format String can be only "D", "d", "N", "n", "P", "p", "B",
"b", "X" or "x".
//       at System.Guid.ParseExact(String input, String format)
//       at Example.Main()
```

This exception also results from a coding error. To correct it, call a parsing method that doesn't require a precise format, such as DateTime.Parse or Guid.Parse, or substitute a valid format string. The following example corrects the error by calling the Guid.Parse method.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string guidString = "ba748d5c-ae5f-4cca-84e5-1ac5291c38cb";
        Console.WriteLine(Guid.Parse(guidString));
```

```
    }
}
// The example displays the following output:
//    ba748d5c-ae5f-4cca-84e5-1ac5291c38cb
```

- One or more of the indexes of the format items in a composite format string is greater than the indexes of the items in the object list or parameter array. In the following example, the largest index of a format item in the format string is 3. Because the indexes of items in the object list are zero-based, this format string would require the object list to have four items. Instead, it has only three, `dat`, `temp`, and `scale`, so the code results in a FormatException exception at run time:.

C#

```
using System;

public class Example
{
   public enum TemperatureScale
   { Celsius, Fahrenheit, Kelvin }

   public static void Main()
   {
      String info = GetCurrentTemperature();
      Console.WriteLine(info);
   }

   private static String GetCurrentTemperature()
   {
      DateTime dat = DateTime.Now;
      Decimal temp = 20.6m;
      TemperatureScale scale = TemperatureScale.Celsius;
      String result;

      result = String.Format("At {0:t} on {1:D}, the temperature is
{2:F1} {3:G}",
                             dat, temp, scale);
      return result;
   }
}
// The example displays output like the following:
//    Unhandled Exception: System.FormatException: Format specifier was
invalid.
//       at System.Number.FormatDecimal(Decimal value, String format,
NumberFormatInfo info)
//       at System.Decimal.ToString(String format, IFormatProvider
provider)
//       at System.Text.StringBuilder.AppendFormat(IFormatProvider
provider, String format, Object[] args)
//       at System.String.Format(IFormatProvider provider, String
```

```
format, Object[] args)
//       at Example.Main()
```

In this case, the FormatException exception is a result of developer error. It should be corrected rather than handled in a `try/catch` block by making sure that each item in the object list corresponds to the index of a format item. To correct this example, change the index of the second format item to refer to the `dat` variable, and decrement the index of each subsequent format item by one.

C#

```
using System;

public class Example
{
   public enum TemperatureScale
   { Celsius, Fahrenheit, Kelvin }

   public static void Main()
   {
      String info = GetCurrentTemperature();
      Console.WriteLine(info);
   }

   private static String GetCurrentTemperature()
   {
      DateTime dat = DateTime.Now;
      Decimal temp = 20.6m;
      TemperatureScale scale = TemperatureScale.Celsius;
      String result;

      result = String.Format("At {0:t} on {0:D}, the temperature is
{1:F1} {2:G}",
                             dat, temp, scale);
      return result;
   }
}
// The example displays output like the following:
//    At 10:40 AM on Wednesday, June 04, 2014, the temperature is 20.6
Celsius
```

- The composite format string isn't well-formed. When this happens, the FormatException exception is always a result of developer error. It should be corrected rather than handled in a `try/catch` block.

  Trying to include literal braces in a string, as the following example does, will throw the exception.

C#

```
result = String.Format("The text has {0} '{' characters and {1} '}'
characters.",
                       nOpen, nClose);
```

The recommended technique for including literal braces in a composite format string is to include them in the object list and use format items to insert them into the result string. For example, you can modify the previous composite format string as shown here.

```C#
string result;
int nOpen = 1;
int nClose = 2;
result = String.Format("The text has {0} '{{' characters and {1} '}}'
characters.",
                       nOpen, nClose);
Console.WriteLine(result);
```

The exception is also thrown if your format string contains a typo. The following call to the String.Format method omits a closing brace and pairs an opening brace with a closing bracket.

```C#
int n1 = 10;
int n2 = 20;
String result = String.Format("{0 + {1] = {2}",
                              n1, n2, n1 + n2);
```

To correct the error, ensure that all opening and closing braces correspond.

```C#
String result = String.Format("{0} + {1} = {2}",
                              n1, n2, n1 + n2);
```

- You've supplied the object list in a composite formatting method as a strongly typed parameter array, and the FormatException exception indicates that the index of one or more format items exceeds the number of arguments in the object list. This occurs because no explicit conversion between array types exists, so instead the compiler treats the array as a single argument rather than as a parameter array. For example, the following call to the Console.WriteLine(String, Object[]) method

throws a FormatException exception, although the highest index of the format items is 3, and the parameter array of type Int32 has four elements.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Random rnd = new Random();
      int[]  numbers = new int[4];
      int total = 0;
      for (int ctr = 0; ctr <= 2; ctr++) {
         int number = rnd.Next(1001);
         numbers[ctr] = number;
         total += number;
      }
      numbers[3] = total;
      Console.WriteLine("{0} + {1} + {2} = {3}", numbers);
   }
}
// The example displays the following output:
//    Unhandled Exception:
//    System.FormatException:
//       Index (zero based) must be greater than or equal to zero and
less than the size of the argument list.
//       at System.Text.StringBuilder.AppendFormat(IFormatProvider
provider, String format, Object[] args)
//       at System.IO.TextWriter.WriteLine(String format, Object arg0)
//       at System.IO.TextWriter.SyncTextWriter.WriteLine(String
format, Object arg0)
//       at Example.Main()
```

Instead of handling this exception, you should eliminate its cause. Because neither Visual Basic nor C# can convert an integer array to an object array, you have to perform the conversion yourself before calling the composite formatting method. The following example provides one implementation.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Random rnd = new Random();
```

```
        int[]  numbers = new int[4];
        int total = 0;
        for (int ctr = 0; ctr <= 2; ctr++) {
           int number = rnd.Next(1001);
           numbers[ctr] = number;
           total += number;
        }
        numbers[3] = total;
        object[] values = new object[numbers.Length];
        numbers.CopyTo(values, 0);
        Console.WriteLine("{0} + {1} + {2} = {3}", values);
    }
}
// The example displays output like the following:
//         477 + 956 + 901 = 2334
```

FormatException uses the HRESULT COR_E_FORMAT, which has the value 0x80131537.

The FormatException class derives from Exception and adds no unique members. For a list of initial property values for an instance of FormatException, see the FormatException constructors.

Constructors

Expand table

FormatException()	Initializes a new instance of the FormatException class.
FormatException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the FormatException class with serialized data.
FormatException(String)	Initializes a new instance of the FormatException class with a specified error message.
FormatException(String, Exception)	Initializes a new instance of the FormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception.

	(Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object.

	(Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

FormatException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the FormatException class.

Overloads

Expand table

FormatException()	Initializes a new instance of the FormatException class.
FormatException(String)	Initializes a new instance of the FormatException class with a specified error message.
FormatException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the FormatException class with serialized data.
FormatException(String, Exception)	Initializes a new instance of the FormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

FormatException()

Initializes a new instance of the FormatException class.

C#

```
public FormatException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Invalid format." This message

takes into account the current system culture.

The following table shows the initial property values for an instance of FormatException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FormatException(String)

Initializes a new instance of the FormatException class with a specified error message.

C#

```
public FormatException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of FormatException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

FormatException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the FormatException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected FormatException (System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

FormatException(String, Exception)

Initializes a new instance of the FormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public FormatException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of FormatException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a composite format string, along with the arguments to be formatted.

C#

```
public abstract class FormattableString : IFormattable
```

Inheritance Object → FormattableString

Implements IFormattable

Remarks

A composite format string consists of fixed text intermixed with indexed placeholders, called format items, that correspond to the objects in the list. The formatting operation yields a result string that consists of the original fixed text intermixed with the string representation of the objects in the list. Composite formatting is supported by methods such as String.Format, Console.WriteLine, and StringBuilder.AppendFormat. For more information on composite formatting, see Composite Formatting.

A FormattableString instance may result from an interpolated string in C# or Visual Basic.

Constructors

Expand table

FormattableString()	Instantiates a new instance of the FormattableString class.

Properties

Expand table

ArgumentCount	Gets the number of arguments to be formatted.
Format	Returns the composite format string.

Methods

Expand table

CurrentCulture(FormattableString)	Returns a result string in which arguments are formatted by using the conventions of the current culture.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetArgument(Int32)	Returns the argument at the specified index position.
GetArguments()	Returns an object array that contains one or more objects to format.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
Invariant(FormattableString)	Returns a result string in which arguments are formatted by using the conventions of the invariant culture.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of the current culture.
ToString(IFormatProvider)	Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of a specified culture.

Explicit Interface Implementations

Expand table

IFormattable.ToString(String, IFormat	Returns the string that results from formatting the format string along with its arguments by using the formatting conventions of a

Provider)	specified culture.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Composite Formatting
- Formatting Types in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Instantiates a new instance of the FormattableString class.

C#

```
protected FormattableString ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString.ArgumentCount Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of arguments to be formatted.

C#

```
public abstract int ArgumentCount { get; }
```

Property Value

Int32

The number of arguments to be formatted.

Remarks

The arguments can be retrieved from the object array returned by the GetArguments method, or they can be retrieved individually by index number from the GetArgument method. The indexes of individual arguments range from zero to ArgumentCount - 1.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

FormattableString.Format Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the composite format string.

```
C#
public abstract string Format { get; }
```

Property Value

String

The composite format string.

Remarks

A composite format string consists of zero or more runs of fixed text intermixed with one or more format items, which are indicated by an index number delimited with brackets (for example, `{0}`). The index of each format item corresponds to an argument in an object list that follows the composite format string. For example, in the following statement:

```
C#
String.Format("Name = {0}, hours = {1:hh}", name, DateTime.Now);
```

the composite format string is `"Name = {0}, hours = {1:hh}"`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Composite Formatting

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString.CurrentCulture(FormattableString) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a result string in which arguments are formatted by using the conventions of the current culture.

```
C#
public static string CurrentCulture (FormattableString formattable);
```

Parameters

`formattable` FormattableString

The string to be formatted.

Returns

String

The string that results from formatting the current instance by using the conventions of the current culture.

Remarks

This static method may be imported in C# or F# by:

```
C#
using static System.FormattableString;
```

``fsharp open type System.FormattableString

```
Within the scope of that import directive, an interpolated string may be
formatted in the current culture by writing, for example:
```csharp
    CurrentCulture($"{{ lat = {latitude}; lon = {longitude} }}");
```

F#

```
    CurrentCulture $"{{ lat = {latitude}; lon = {longitude} }}"
```

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback
```

FormattableString.GetArgument(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the argument at the specified index position.

```
C#
public abstract object? GetArgument (int index);
```

Parameters

`index` Int32

The index of the argument. Its value can range from zero to one less than the value of ArgumentCount.

Returns

Object

The argument.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

FormattableString.GetArguments Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an object array that contains one or more objects to format.

C#

```
public abstract object?[] GetArguments ();
```

Returns

Object[]

An object array that contains one or more objects to format.

Remarks

Clients should not modify the contents of the array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

FormattableString.Invariant(FormattableString) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a result string in which arguments are formatted by using the conventions of the invariant culture.

```
C#
public static string Invariant (FormattableString formattable);
```

Parameters

formattable FormattableString

The object to convert to a result string.

Returns

String

The string that results from formatting the current instance by using the conventions of the invariant culture.

Exceptions

ArgumentNullException

`formattable` is `null`.

Remarks

This static method can be imported in C# by the following statement:

```
C#
```

```
using static System.FormattableString;
```

Within the scope that `using static` directive, an interpolated string may be formatted using the conventions of the invariant culture by code such as the following:

```C#
Invariant($"{{ lat = {latitude}; lon = {longitude} }}")
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string that results from formatting the composite format string along with its arguments.

Overloads

Expand table

ToString()	Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of the current culture.
ToString(IFormatProvider)	Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of a specified culture.

ToString()

Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of the current culture.

C#

```
public override string ToString ();
```

Returns

String

A result string formatted by using the conventions of the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Returns the string that results from formatting the composite format string along with its arguments by using the formatting conventions of a specified culture.

C#

```
public abstract string ToString (IFormatProvider? formatProvider);
```

Parameters

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

Returns

String

A result string formatted by using the conventions of `formatProvider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FormattableString.IFormattable.ToString(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string that results from formatting the format string along with its arguments by using the formatting conventions of a specified culture.

```
C#

string IFormattable.ToString (string ignored, IFormatProvider
formatProvider);
```

Parameters

ignored String

A string. This argument is ignored.

formatProvider IFormatProvider

An object that provides culture-specific formatting information.

Returns

String

A string formatted using the conventions of the `formatProvider` parameter.

Implements

ToString(String, IFormatProvider)

Remarks

This implementation calls the derived class' ToString(IFormatProvider) implementation.

This member is an explicit interface implementation. It can be used only when the FormattableString instance is cast to an IFormattable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

FtpStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the File Transfer Protocol (FTP) scheme.

C#

```
public class FtpStyleUriParser : UriParser
```

Inheritance Object → UriParser → FtpStyleUriParser

Constructors

Expand table

FtpStyleUriParser()	Creates a customizable parser based on the File Transfer Protocol (FTP) scheme.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)

InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

FtpStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a customizable parser based on the File Transfer Protocol (FTP) scheme.

C#

```
public FtpStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has no parameters and returns a value of the type specified by the `TResult` parameter.

```
C#

public delegate TResult Func<out TResult>();
```

Type Parameters

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Examples

The following example demonstrates how to use a delegate that takes no parameters. This code creates a generic class named `LazyValue` that has a field of type Func<TResult>. This delegate field can store a reference to any function that returns a value of the type that corresponds to the type parameter of the `LazyValue` object. The `LazyValue` type also has a `Value` property that executes the function (if it has not already been executed) and returns the resulting value.

The example creates two methods and instantiates two `LazyValue` objects with lambda expressions that call these methods. The lambda expressions do not take parameters

because they just need to call a method. As the output shows, the two methods are executed only when the value of each `LazyValue` object is retrieved.

C#

```
using System;

static class Func1
{
   public static void Main()
   {
      // Note that each lambda expression has no parameters.
      LazyValue<int> lazyOne = new LazyValue<int>(() => ExpensiveOne());
      LazyValue<long> lazyTwo = new LazyValue<long>(() =>
ExpensiveTwo("apple"));

      Console.WriteLine("LazyValue objects have been created.");

      // Get the values of the LazyValue objects.
      Console.WriteLine(lazyOne.Value);
      Console.WriteLine(lazyTwo.Value);
   }

   static int ExpensiveOne()
   {
      Console.WriteLine("\nExpensiveOne() is executing.");
      return 1;
   }

   static long ExpensiveTwo(string input)
   {
      Console.WriteLine("\nExpensiveTwo() is executing.");
      return (long)input.Length;
   }
}

class LazyValue<T> where T : struct
{
   private Nullable<T> val;
   private Func<T> getValue;

   // Constructor.
   public LazyValue(Func<T> func)
   {
      val = null;
      getValue = func;
   }

   public T Value
   {
      get
      {
         if (val == null)
            // Execute the delegate.
```

```
            val = getValue();
        return (T)val;
    }
  }
}
/* The example produces the following output:

    LazyValue objects have been created.

    ExpensiveOne() is executing.
    1

    ExpensiveTwo() is executing.
    5
*/
```

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have no parameters and must return a value.

> **Note**
>
> To reference a method that has no parameters and returns `void` (`unit`, in F#) (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the **Action** delegate instead.

When you use the Func<TResult> delegate, you do not have to explicitly define a delegate that encapsulates a parameterless method. For example, the following code explicitly declares a delegate named `WriteMethod` and assigns a reference to the `OutputTarget.SendToFile` instance method to its delegate instance.

C#

```
using System;
using System.IO;

delegate bool WriteMethod();

public class TestDelegate
{
   public static void Main()
   {
      OutputTarget output = new OutputTarget();
```

```
        WriteMethod methodCall = output.SendToFile;
        if (methodCall())
           Console.WriteLine("Success!");
        else
           Console.WriteLine("File write operation failed.");
    }
}

public class OutputTarget
{
    public bool SendToFile()
    {
        try
        {
           string fn = Path.GetTempFileName();
           StreamWriter sw = new StreamWriter(fn);
           sw.WriteLine("Hello, World!");
           sw.Close();
           return true;
        }
        catch
        {
           return false;
        }
    }
}
```

The following example simplifies this code by instantiating the Func<TResult> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;
using System.IO;

public class TestDelegate
{
    public static void Main()
    {
        OutputTarget output = new OutputTarget();
        Func<bool> methodCall = output.SendToFile;
        if (methodCall())
           Console.WriteLine("Success!");
        else
           Console.WriteLine("File write operation failed.");
    }
}

public class OutputTarget
{
    public bool SendToFile()
    {
        try
```

```
        {
            string fn = Path.GetTempFileName();
            StreamWriter sw = new StreamWriter(fn);
            sw.WriteLine("Hello, World!");
            sw.Close();
            return true;
        }
        catch
        {
            return false;
        }
    }
}
```

You can use the Func<TResult> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;
using System.IO;

public class Anonymous
{
    public static void Main()
    {
        OutputTarget output = new OutputTarget();
        Func<bool> methodCall = delegate() { return output.SendToFile(); };
        if (methodCall())
            Console.WriteLine("Success!");
        else
            Console.WriteLine("File write operation failed.");
    }
}

public class OutputTarget
{
    public bool SendToFile()
    {
        try
        {
            string fn = Path.GetTempFileName();
            StreamWriter sw = new StreamWriter(fn);
            sw.WriteLine("Hello, World!");
            sw.Close();
            return true;
        }
        catch
        {
            return false;
        }
```

```
    }
}
```

You can also assign a lambda expression to a Func<T,TResult> delegate, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (VB), Lambda Expressions (C#), and Lambda Expressions (F#).)

C#

```
using System;
using System.IO;

public class Anonymous
{
   public static void Main()
   {
      OutputTarget output = new OutputTarget();
      Func<bool> methodCall = () => output.SendToFile();
      if (methodCall())
         Console.WriteLine("Success!");
      else
         Console.WriteLine("File write operation failed.");
   }
}

public class OutputTarget
{
   public bool SendToFile()
   {
      try
      {
         string fn = Path.GetTempFileName();
         StreamWriter sw = new StreamWriter(fn);
         sw.WriteLine("Hello, World!");
         sw.Close();
         return true;
      }
      catch
      {
         return false;
      }
   }
}
```

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the System.Linq namespace have `Func` parameters, you can pass these methods a lambda expression without explicitly instantiating a `Func` delegate.

If you have an expensive computation that you want to execute only if the result is actually needed, you can assign the expensive function to a Func<TResult> delegate. The execution of the function can then be delayed until a property that accesses the value is used in an expression. The example in the next section demonstrates how to do this.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lambda Expressions (C# Programming Guide)
- Lambda Expressions: The fun Keyword (F#)
- Lambda Expressions
- Delegates (C# Programming Guide)
- Delegates (F#)
- Delegates in Visual Basic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Func<T,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has one parameter and returns a value of the type specified by the `TResult` parameter.

```
C#
public delegate TResult Func<in T,out TResult>(T arg);
```

Type Parameters

`T`

The type of the parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg` T

The parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Examples

The following example demonstrates how to declare and use a Func<T,TResult> delegate. This example declares a Func<T,TResult> variable and assigns it a lambda expression that converts the characters in a string to uppercase. The delegate that encapsulates this method is subsequently passed to the Enumerable.Select method to change the strings in an array of strings to uppercase.

C#

```
// Declare a Func variable and assign a lambda expression to the
// variable. The method takes a string and converts it to uppercase.
Func<string, string> selector = str => str.ToUpper();

// Create an array of strings.
string[] words = { "orange", "apple", "Article", "elephant" };
// Query the array and select strings according to the selector method.
IEnumerable<String> aWords = words.Select(selector);

// Output the results to the console.
foreach (String word in aWords)
    Console.WriteLine(word);

/*
This code example produces the following output:

ORANGE
APPLE
ARTICLE
ELEPHANT

*/
```

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have one parameter that is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has one parameter and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T>**

delegate instead.

When you use the Func<T,TResult> delegate, you do not have to explicitly define a delegate that encapsulates a method with a single parameter. For example, the following code explicitly declares a delegate named `ConvertMethod` and assigns a reference to the `UppercaseString` method to its delegate instance.

C#

```
using System;

delegate string ConvertMethod(string inString);

public class DelegateExample
{
   public static void Main()
   {
      // Instantiate delegate to reference UppercaseString method
      ConvertMethod convertMeth = UppercaseString;
      string name = "Dakota";
      // Use delegate instance to call UppercaseString method
      Console.WriteLine(convertMeth(name));
   }

   private static string UppercaseString(string inputString)
   {
      return inputString.ToUpper();
   }
}
```

The following example simplifies this code by instantiating the Func<T,TResult> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
// Instantiate delegate to reference UppercaseString method
Func<string, string> convertMethod = UppercaseString;
string name = "Dakota";
// Use delegate instance to call UppercaseString method
Console.WriteLine(convertMethod(name));

string UppercaseString(string inputString)
{
   return inputString.ToUpper();
}

// This code example produces the following output:
```

```
//
//    DAKOTA
```

You can also use the Func<T,TResult> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
 Func<string, string> convert = delegate(string s)
    { return s.ToUpper();};

 string name = "Dakota";
 Console.WriteLine(convert(name));

// This code example produces the following output:
//
//    DAKOTA
```

You can also assign a lambda expression to a Func<T,TResult> delegate, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (VB), Lambda Expressions (C#) and Lambda Expressions (F#).)

C#

```
Func<string, string> convert = s => s.ToUpper();

string name = "Dakota";
Console.WriteLine(convert(name));

// This code example produces the following output:
//
//    DAKOTA
```

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the System.Linq namespace have Func<T,TResult> parameters, you can pass these methods a lambda expression without explicitly instantiating a Func<T,TResult> delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lambda Expressions (C# Programming Guide)
- Lambda Expressions: The fun Keyword (F#)
- Lambda Expressions (Visual Basic)
- Delegates (C# Programming Guide)
- Delegates (F#)
- Delegates (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has two parameters and returns a value of the type specified by the `TResult` parameter.

```
C#
public delegate TResult Func<in T1,in T2,out TResult>(T1 arg1, T2 arg2);
```

Type Parameters

`T1`

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T2`

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

arg2 T2

The second parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Examples

The following example demonstrates how to declare and use a Func<T1,T2,TResult> delegate. This example declares a Func<T1,T2,TResult> variable and assigns it a lambda expression that takes a String value and an Int32 value as parameters. The lambda expression returns `true` if the length of the String parameter is equal to the value of the Int32 parameter. The delegate that encapsulates this method is subsequently used in a query to filter strings in an array of strings.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Func3Example
{
   public static void Main()
   {
      Func<String, int, bool> predicate = (str, index) => str.Length == index;

      String[] words = { "orange", "apple", "Article", "elephant", "star", "and" };
      IEnumerable<String> aWords = words.Where(predicate).Select(str => str);

      foreach (String word in aWords)
         Console.WriteLine(word);
   }
}
```

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must

correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have two parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has two parameters and returns `void` (`unit` in F#) (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2>** delegate instead.

When you use the Func<T1,T2,TResult> delegate you do not have to explicitly define a delegate that encapsulates a method with two parameters. For example, the following code explicitly declares a delegate named `ExtractMethod` and assigns a reference to the `ExtractWords` method to its delegate instance.

C#

```
using System;

delegate string[] ExtractMethod(string stringToManipulate, int maximum);

public class DelegateExample
{
   public static void Main()
   {
      // Instantiate delegate to reference ExtractWords method
      ExtractMethod extractMeth = ExtractWords;
      string title = "The Scarlet Letter";
      // Use delegate instance to call ExtractWords method and display
result
      foreach (string word in extractMeth(title, 5))
         Console.WriteLine(word);
   }

   private static string[] ExtractWords(string phrase, int limit)
   {
      char[] delimiters = new char[] {' '};
      if (limit > 0)
         return phrase.Split(delimiters, limit);
      else
         return phrase.Split(delimiters);
   }
}
```

The following example simplifies this code by instantiating a Func<T1,T2,TResult> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;

public class GenericFunc
{
   public static void Main()
   {
      // Instantiate delegate to reference ExtractWords method
      Func<string, int, string[]> extractMethod = ExtractWords;
      string title = "The Scarlet Letter";
      // Use delegate instance to call ExtractWords method and display
result
      foreach (string word in extractMethod(title, 5))
         Console.WriteLine(word);
   }

   private static string[] ExtractWords(string phrase, int limit)
   {
      char[] delimiters = new char[] {' '};
      if (limit > 0)
         return phrase.Split(delimiters, limit);
      else
         return phrase.Split(delimiters);
   }
}
```

You can use the Func<T1,T2,TResult> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;

public class Anonymous
{
   public static void Main()
   {
      Func<string, int, string[]> extractMeth = delegate(string s, int i)
         { char[] delimiters = new char[] {' '};
           return i > 0 ? s.Split(delimiters, i) : s.Split(delimiters);
         };

      string title = "The Scarlet Letter";
      // Use Func instance to call ExtractWords method and display result
      foreach (string word in extractMeth(title, 5))
         Console.WriteLine(word);
   }
}
```

You can also assign a lambda expression to a Func<T1,T2,TResult> delegate, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (VB), Lambda Expressions (C#) and Lambda Expressions (F#).)

C#

```
using System;

public class LambdaExpression
{
   public static void Main()
   {
      char[] separators = new char[] {' '};
      Func<string, int, string[]> extract = (s, i) =>
           i > 0 ? s.Split(separators, i) : s.Split(separators) ;

      string title = "The Scarlet Letter";
      // Use Func instance to call ExtractWords method and display result
      foreach (string word in extract(title, 5))
         Console.WriteLine(word);
   }
}
```

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the System.Linq namespace have Func<T1,T2,TResult> parameters, you can pass these methods a lambda expression without explicitly instantiating a Func<T1,T2,TResult> delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lambda Expressions (C# Programming Guide)
- Lambda Expressions: The fun Keyword (F#)
- Lambda Expressions
- Delegates (C# Programming Guide)
- Delegates (F#)
- Delegates in Visual Basic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has three parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,out TResult>(T1 arg1, T2
arg2, T3 arg3);
```

Type Parameters

`T1`

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T2`

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T3`

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.

This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Examples

The following example demonstrates how to declare and use a Func<T1,T2,TResult> delegate. This example declares a Func<T1,T2,TResult> variable and assigns it a lambda expression that takes a String value and an Int32 value as parameters. The lambda expression returns `true` if the length of the String parameter is equal to the value of the Int32 parameter. The delegate that encapsulates this method is subsequently used in a query to filter strings in an array of strings.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Func3Example
{
   public static void Main()
   {
      Func<String, int, bool> predicate = (str, index) => str.Length ==
index;

      String[] words = { "orange", "apple", "Article", "elephant", "star",
"and" };
```

```
      IEnumerable<String> aWords = words.Where(predicate).Select(str =>
str);

      foreach (String word in aWords)
         Console.WriteLine(word);
   }
}
```

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have three parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has three parameters and returns `void` (`unit` in F#) (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3>** delegate instead.

When you use the Func<T1,T2,T3,TResult> delegate, you do not have to explicitly define a delegate that encapsulates a method with three parameters. For example, the following code explicitly declares a generic delegate named `ParseNumber` and assigns a reference to the Parse method to its delegate instance.

C#

```
using System;
using System.Globalization;

delegate T ParseNumber<T>(string input, NumberStyles styles,
                          IFormatProvider provider);

public class DelegateExample
{
   public static void Main()
   {
      string numericString = "-1,234";
      ParseNumber<int> parser = int.Parse;
      Console.WriteLine(parser(numericString,
                               NumberStyles.Integer | NumberStyles.AllowThousands,
                               CultureInfo.InvariantCulture));
```

```
    }
}
```

The following example simplifies this code by instantiating the Func<T1,T2,T3,TResult> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;
using System.Globalization;

public class GenericFunc
{
   public static void Main()
   {
      string numericString = "-1,234";
      Func<string, NumberStyles, IFormatProvider, int> parser = int.Parse;
      Console.WriteLine(parser(numericString,
                               NumberStyles.Integer | NumberStyles.AllowThousands,
                               CultureInfo.InvariantCulture));
   }
}
```

You can use the Func<T1,T2,T3,TResult> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;
using System.Globalization;

public class Anonymous
{
   public static void Main()
   {
      string numericString = "-1,234";
      Func<string, NumberStyles, IFormatProvider, int> parser =
           delegate(string s, NumberStyles sty, IFormatProvider p)
           { return int.Parse(s, sty, p); };
      Console.WriteLine(parser(numericString,
                               NumberStyles.Integer | NumberStyles.AllowThousands,
                               CultureInfo.InvariantCulture));
   }
}
```

You can also assign a lambda expression to a Func<T1,T2,T3,TResult> delegate, as the following example illustrates. (For an introduction to lambda expressions, see Lambda

Expressions (VB), Lambda Expressions (C#) and Lambda Expressions (F#).)

C#

```
using System;
using System.Globalization;

public class LambdaExpression
{
   public static void Main()
   {
      string numericString = "-1,234";
      Func<string, NumberStyles, IFormatProvider, int> parser = (s, sty, p)
                   => int.Parse(s, sty, p);
      Console.WriteLine(parser(numericString,
                       NumberStyles.Integer | NumberStyles.AllowThousands,
                       CultureInfo.InvariantCulture));
   }
}
```

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the System.Linq namespace have `Func` parameters, you can pass these methods a lambda expression without explicitly instantiating a `Func` delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lambda Expressions (C# Programming Guide)
- Lambda Expressions
- Lambda Expressions: The fun Keyword (F#)
- Delegates (C# Programming Guide)
- Delegates (F#)
- Delegates in Visual Basic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has four parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,out TResult>(T1 arg1,
T2 arg2, T3 arg3, T4 arg4);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Examples

The following example demonstrates how to declare and use a Func<T1,T2,TResult> delegate. This example declares a Func<T1,T2,TResult> variable and assigns it a lambda expression that takes a String value and an Int32 value as parameters. The lambda expression returns `true` if the length of the String parameter is equal to the value of the Int32 parameter. The delegate that encapsulates this method is subsequently used in a query to filter strings in an array of strings.

```
C#
```

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Func3Example
{
   public static void Main()
   {
      Func<String, int, bool> predicate = (str, index) => str.Length ==
index;

      String[] words = { "orange", "apple", "Article", "elephant", "star",
"and" };
      IEnumerable<String> aWords = words.Where(predicate).Select(str =>
str);

      foreach (String word in aWords)
         Console.WriteLine(word);
   }
}
```

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have four parameters, each of which is passed to it by value, and that it must return a value.

> ⓘ **Note**
>
> To reference a method that has four parameters and returns `void` (`unit` in F#) (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4>** delegate instead.

When you use the Func<T1,T2,T3,T4,TResult> delegate, you do not have to explicitly define a delegate that encapsulates a method with four parameters. For example, the following code explicitly declares a generic delegate named `Searcher` and assigns a reference to the IndexOf method to its delegate instance.

C#

```
using System;

delegate int Searcher(string searchString, int start, int count,
```

```
                          StringComparison type);

public class DelegateExample
{
   public static void Main()
   {
      string title = "The House of the Seven Gables";
      int position = 0;
      Searcher finder = title.IndexOf;
      do
      {
         int characters = title.Length - position;
         position = finder("the", position, characters,
                          StringComparison.InvariantCultureIgnoreCase);
         if (position >= 0)
         {
            position++;
            Console.WriteLine("'The' found at position {0} in {1}.",
                              position, title);
         }
      } while (position > 0);
   }
}
```

The following example simplifies this code by instantiating the Func<T1,T2,T3,T4,TResult> delegate instead of explicitly defining a new delegate and assigning a named method to it.

C#

```
using System;

public class DelegateExample
{
   public static void Main()
   {
      string title = "The House of the Seven Gables";
      int position = 0;
      Func<string, int, int, StringComparison, int> finder = title.IndexOf;
      do
      {
         int characters = title.Length - position;
         position = finder("the", position, characters,
                          StringComparison.InvariantCultureIgnoreCase);
         if (position >= 0)
         {
            position++;
            Console.WriteLine("'The' found at position {0} in {1}.",
                              position, title);
         }
      } while (position > 0);
```

```
    }
}
```

You can use the Func<T1,T2,T3,T4,TResult> delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see Anonymous Methods.)

C#

```
using System;

public class DelegateExample
{
   public static void Main()
   {
      string title = "The House of the Seven Gables";
      int position = 0;
      Func<string, int, int, StringComparison, int> finder =
           delegate(string s, int pos, int chars, StringComparison type)
           { return title.IndexOf(s, pos, chars, type); };
      do
      {
         int characters = title.Length - position;
         position = finder("the", position, characters,
                           StringComparison.InvariantCultureIgnoreCase);
         if (position >= 0)
         {
            position++;
            Console.WriteLine("'The' found at position {0} in {1}.",
                              position, title);
         }
      } while (position > 0);
   }
}
```

You can also assign a lambda expression to a Func<T1,T2,TResult> delegate, as the following example illustrates. (For an introduction to lambda expressions, see Lambda Expressions (VB), Lambda Expressions (C#) and Lambda Expressions (F#).)

C#

```
using System;

public class DelegateExample
{
   public static void Main()
   {
      string title = "The House of the Seven Gables";
      int position = 0;
      Func<string, int, int, StringComparison, int> finder =
```

```
            (s, pos, chars, type) => title.IndexOf(s, pos, chars, type);
        do
        {
           int characters = title.Length - position;
           position = finder("the", position, characters,
                             StringComparison.InvariantCultureIgnoreCase);
           if (position >= 0)
           {
              position++;
              Console.WriteLine("'The' found at position {0} in {1}.",
                                position, title);
           }
        } while (position > 0);
    }
}
```

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the System.Linq namespace have `Func` parameters, you can pass these methods a lambda expression without explicitly instantiating a `Func` delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lambda Expressions (C# Programming Guide)

- Lambda Expressions: The fun Keyword (F#)
- Lambda Expressions
- Delegates (C# Programming Guide)
- Delegates (F#)
- Delegates in Visual Basic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has five parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.

This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have five parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has five parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Func<T1,T2,T3,T4,T5,T6,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has six parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,out
TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

arg6 T6

The sixth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have six parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has six parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has seven parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,out
TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have seven parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has seven parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has eight parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7
arg7, T8 arg8);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have eight parameters, each of which is passed to it by value, and that it must return a value.

Note

To reference a method that has eight parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has nine parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,in T9,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6,
T7 arg7, T8 arg8, T9 arg9);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that

the encapsulated method must have nine parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has nine parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 10 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in
T8,in T9,in T10,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6
arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 10 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 10 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 11 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T4

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 11 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 11 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 12 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T4`

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 12 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 12 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 13 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T4`

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 13 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 13 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 14 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,in T14,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13, T14 arg14);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T4`

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T14`

The type of the fourteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

`arg14` T14

The fourteenth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 14 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 14 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 15 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,in T14,in T15,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13, T14 arg14, T15 arg15);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T4`

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T14`

The type of the fourteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T15`

The type of the fifteenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

arg14 T14

The fourteenth parameter of the method that this delegate encapsulates.

arg15 T15

The fifteenth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 15 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 15 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Encapsulates a method that has 16 parameters and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T1,in T2,in T3,in T4,in T5,in T6,in T7,in T8,in T9,in T10,in T11,in T12,in T13,in T14,in T15,in T16,out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7, T8 arg8, T9 arg9, T10 arg10, T11 arg11, T12 arg12, T13 arg13, T14 arg14, T15 arg15, T16 arg16);
```

Type Parameters

T1

The type of the first parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T2

The type of the second parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

T3

The type of the third parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T4`

The type of the fourth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T5`

The type of the fifth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T6`

The type of the sixth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T7`

The type of the seventh parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T8`

The type of the eighth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T9`

The type of the ninth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T10`

The type of the tenth parameter of the method that this delegate encapsulates.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance,

see Covariance and Contravariance in Generics.

`T11`

The type of the eleventh parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T12`

The type of the twelfth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T13`

The type of the thirteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T14`

The type of the fourteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T15`

The type of the fifteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`T16`

The type of the sixteenth parameter of the method that this delegate encapsulates. This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

`TResult`

The type of the return value of the method that this delegate encapsulates.

This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

`arg1` T1

The first parameter of the method that this delegate encapsulates.

`arg2` T2

The second parameter of the method that this delegate encapsulates.

`arg3` T3

The third parameter of the method that this delegate encapsulates.

`arg4` T4

The fourth parameter of the method that this delegate encapsulates.

`arg5` T5

The fifth parameter of the method that this delegate encapsulates.

`arg6` T6

The sixth parameter of the method that this delegate encapsulates.

`arg7` T7

The seventh parameter of the method that this delegate encapsulates.

`arg8` T8

The eighth parameter of the method that this delegate encapsulates.

`arg9` T9

The ninth parameter of the method that this delegate encapsulates.

`arg10` T10

The tenth parameter of the method that this delegate encapsulates.

`arg11` T11

The eleventh parameter of the method that this delegate encapsulates.

`arg12` T12

The twelfth parameter of the method that this delegate encapsulates.

`arg13` T13

The thirteenth parameter of the method that this delegate encapsulates.

`arg14` T14

The fourteenth parameter of the method that this delegate encapsulates.

`arg15` T15

The fifteenth parameter of the method that this delegate encapsulates.

`arg16` T16

The sixteenth parameter of the method that this delegate encapsulates.

Return Value

TResult

The return value of the method that this delegate encapsulates.

Remarks

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have 16 parameters, each of which is passed to it by value, and that it must return a value.

> **Note**
>
> To reference a method that has 16 parameters and returns `void` (or in Visual Basic, that is declared as a `Sub` rather than as a `Function`), use the generic **Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>** delegate instead.

You can also use the Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult> delegate with anonymous methods and lambda expressions.

The underlying type of a lambda expression is one of the generic `Func` delegates. This makes it possible to pass a lambda expression as a parameter without explicitly

assigning it to a delegate.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Controls the system garbage collector, a service that automatically reclaims unused memory.

C#

```
public static class GC
```

Inheritance Object → GC

Examples

The following example uses several GC methods to get generation and memory information about a block of unused objects and print it to the console. The unused objects are then collected, and the resulting memory totals are displayed.

C#

```
using System;

namespace GCCollectIntExample
{
    class MyGCCollectClass
    {
        private const long maxGarbage = 1000;

        static void Main()
        {
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Determine the maximum number of generations the system
        // garbage collector currently supports.
            Console.WriteLine("The highest generation is {0}",
GC.MaxGeneration);

            myGCCol.MakeSomeGarbage();
```

```
            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            // Determine the best available approximation of the number
        // of bytes currently allocated in managed memory.
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of generation 0 only.
            GC.Collect(0);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of all generations up to and including
2.
            GC.Collect(2);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));
            Console.Read();
        }

        void MakeSomeGarbage()
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
        // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

For more information about this API, see Supplemental API remarks for GC.

Properties

Expand table

MaxGeneration	Gets the maximum number of generations that the system currently supports.

Methods

Expand table

AddMemoryPressure(Int64)	Informs the runtime of a large allocation of unmanaged memory that should be taken into account when scheduling garbage collection.
AllocateArray<T>(Int32, Boolean)	Allocates an array.
AllocateUninitializedArray<T>(Int32, Boolean)	Allocates an array while skipping zero-initialization, if possible.
CancelFullGCNotification()	Cancels the registration of a garbage collection notification.
Collect()	Forces an immediate garbage collection of all generations.
Collect(Int32)	Forces an immediate garbage collection from generation 0 through a specified generation.
Collect(Int32, GCCollectionMode)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value.
Collect(Int32, GCCollectionMode, Boolean)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with a value specifying whether the collection should be blocking.
Collect(Int32, GCCollectionMode, Boolean, Boolean)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with values that specify whether the collection should be blocking and compacting.
CollectionCount(Int32)	Returns the number of times garbage collection has occurred for the specified generation of objects.
EndNoGCRegion()	Ends the no GC region latency mode.
GetAllocatedBytesForCurrentThread()	Gets the total number of bytes allocated to the current thread since the beginning of its lifetime.
GetConfigurationVariables()	Gets the configurations used by the garbage collector.
GetGCMemoryInfo()	Gets garbage collection memory information.

GetGCMemoryInfo(GCKind)	Gets garbage collection memory information.
GetGeneration(Object)	Returns the current generation number of the specified object.
GetGeneration(WeakReference)	Returns the current generation number of the target of a specified weak reference.
GetTotalAllocatedBytes(Boolean)	Gets a count of the bytes allocated over the lifetime of the process. The returned value does not include any native allocations.
GetTotalMemory(Boolean)	Retrieves the heap size excluding fragmentation. For example if the total GC heap size is 100mb and fragmentation, ie, space taken up by free objects, takes up 40mb, this API would report 60mb. A parameter indicates whether this method can wait a short interval before returning, to allow the system to collect garbage and finalize objects.
GetTotalPauseDuration()	Gets the total amount of time paused in GC since the beginning of the process.
KeepAlive(Object)	References the specified object, which makes it ineligible for garbage collection from the start of the current routine to the point where this method is called.
RefreshMemoryLimit()	Instructs the Garbage Collector to reconfigure itself by detecting the various memory limits on the system.
RegisterForFullGCNotification(Int32, Int32)	Specifies that a garbage collection notification should be raised when conditions favor full garbage collection and when the collection has been completed.
RegisterNoGCRegionCallback(Int64, Action)	Registers a callback to be invoked when a certain amount of memory is allocated in the no GC region.
RemoveMemoryPressure(Int64)	Informs the runtime that unmanaged memory has been released and no longer needs to be taken into account when scheduling garbage collection.
ReRegisterForFinalize(Object)	Requests that the system call the finalizer for the specified object for which SuppressFinalize(Object) has previously been called.
SuppressFinalize(Object)	Requests that the common language runtime not call the finalizer for the specified object.
TryStartNoGCRegion(Int64)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available.
TryStartNoGCRegion(Int64, Boolean)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available, and

	controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.
TryStartNoGCRegion(Int64, Int64)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap.
TryStartNoGCRegion(Int64, Int64, Boolean)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap, and controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.
WaitForFullGCApproach()	Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.
WaitForFull GCApproach(Int32)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.
WaitForFull GCApproach(TimeSpan)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.
WaitForFullGCComplete()	Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime has completed.
WaitForFull GCComplete(Int32)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by common language the runtime has completed.
WaitForFull GCComplete(TimeSpan)	Returns the status of a registered notification about whether a blocking garbage collection has completed. May wait indefinitely for a full collection.
WaitForPendingFinalizers()	Suspends the current thread until the thread that is processing the queue of finalizers has emptied that queue.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- GCSettings
- Garbage Collection
- Fundamentals of Garbage Collection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.MaxGeneration Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum number of generations that the system currently supports.

C#

```
public static int MaxGeneration { get; }
```

Property Value

Int32

A value that ranges from zero to the maximum number of supported generations.

Examples

The following example demonstrates how to use the MaxGeneration property to display the largest generation number currently in use.

C#

```
using System;

namespace GCCollectIntExample
{
    class MyGCCollectClass
    {
        private const long maxGarbage = 1000;

        static void Main()
        {
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Determine the maximum number of generations the system
        // garbage collector currently supports.
            Console.WriteLine("The highest generation is {0}",
GC.MaxGeneration);
```

```
            myGCCol.MakeSomeGarbage();

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            // Determine the best available approximation of the number
        // of bytes currently allocated in managed memory.
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of generation 0 only.
            GC.Collect(0);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of all generations up to and including
2.
            GC.Collect(2);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));
            Console.Read();
        }

        void MakeSomeGarbage()
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
        // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

The generation number, or age, of an object is an implementation-defined relative measure of an object's lifespan. The most recently created objects are in generation 0 and the oldest objects are in a generation less than or equal to the generation returned by the MaxGeneration property.

The garbage collector assumes that newer memory is more likely to be eligible for garbage collection than older memory. Therefore, the garbage collector improves its performance by adjusting generation numbers each time it reclaims memory, and the MaxGeneration property value can grow over time.

If object aging is implemented, the MaxGeneration property returns the maximum generation number used by the system; otherwise, this property returns zero.

For this implementation, the value returned by the MaxGeneration property is guaranteed to remain constant for the lifetime of an executing application.

Use the MaxGeneration property to determine the maximum value you can specify when calling the Collect(Int32) method that takes a generation parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetGeneration(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.AddMemoryPressure(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Informs the runtime of a large allocation of unmanaged memory that should be taken into account when scheduling garbage collection.

```
C#
public static void AddMemoryPressure (long bytesAllocated);
```

Parameters

`bytesAllocated` Int64

The incremental amount of unmanaged memory that has been allocated.

Exceptions

ArgumentOutOfRangeException

`bytesAllocated` is less than or equal to 0.

-or-

On a 32-bit computer, `bytesAllocated` is larger than Int32.MaxValue.

Remarks

In determining when to schedule garbage collection, the runtime takes into account how much managed memory is allocated. If a small managed object allocates a large amount of unmanaged memory, the runtime takes into account only the managed memory, and thus underestimates the urgency of scheduling garbage collection. The AddMemoryPressure method informs the runtime of this additional pressure on system memory.

The AddMemoryPressure and RemoveMemoryPressure methods improve performance only for types that exclusively depend on finalizers to release the unmanaged resources.

It's not necessary to use these methods in types that follow the dispose pattern, where finalizers are used to clean up unmanaged resources only in the event that a consumer of the type forgets to call `Dispose`. For more information on object finalization and the dispose pattern, see Cleaning Up Unmanaged Resources.

In the simplest usage pattern, a managed object allocates unmanaged memory in the constructor and releases it in the `Finalize` method. Call the AddMemoryPressure method after allocating the unmanaged memory, and call the RemoveMemoryPressure method after releasing it.

In more complicated scenarios, where the unmanaged memory allocation changes substantially during the lifetime of the managed object, you can call the AddMemoryPressure and RemoveMemoryPressure methods to communicate these incremental changes to the runtime.

> **Caution**
>
> You must ensure that you remove exactly the amount of pressure you add. Failing to do so can adversely affect the performance of the system in applications that run for long periods of time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

GC.AllocateArray<T>(Int32, Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Allocates an array.

C#

```
public static T[] AllocateArray<T> (int length, bool pinned = false);
```

Type Parameters

T

Specifies the type of the array element.

Parameters

length Int32

Specifies the length of the array.

pinned Boolean

Specifies whether the allocated array must be pinned.

Returns

T[]

An array object.

Remarks

If pinned is set to `true`, `T` must not be a reference type or a type that contains object references.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GC.AllocateUninitializedArray<T>(Int32, Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Allocates an array while skipping zero-initialization, if possible.

C#

```
public static T[] AllocateUninitializedArray<T> (int length, bool pinned =
false);
```

Type Parameters

`T`

Specifies the type of the array element.

Parameters

`length` Int32

Specifies the length of the array.

`pinned` Boolean

Specifies whether the allocated array must be pinned.

Returns

T[]

An array object with uninitialized memory.

Remarks

If pinned is set to `true`, `T` must not be a reference type or a type that contains object references.

Skipping zero-initialization is a security risk. The unitialized array can contain invalid valuetype instances or sensitive information created by other parts of the application. The code operating on unitialized arrays should be heavily scrutinized to ensure that the unitialized data is never read.

Skipping zero-initialization using this API only has a material performance benefit for large arrays, such as buffers of several kilobytes or more.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.CancelFullGCNotification Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Cancels the registration of a garbage collection notification.

C#

```
public static void CancelFullGCNotification ();
```

Exceptions

InvalidOperationException

This member is not available when concurrent garbage collection is enabled. See the <gcConcurrent> runtime setting for information about how to disable concurrent garbage collection.

Examples

The following example cancels a garbage collection registration. This example is part of a larger example provided for the Garbage Collection Notifications topic.

C#

```
finalExit = true;
checkForNotify = false;
GC.CancelFullGCNotification();
```

Remarks

This method cancels a garbage collection notification that was registered by using the RegisterForFullGCNotification method. You do not have to call this method before adjusting threshold parameter values in subsequent calls to the RegisterForFullGCNotification method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Garbage Collection Notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.Collect Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Forces garbage collection.

Overloads

Expand table

Collect()	Forces an immediate garbage collection of all generations.
Collect(Int32)	Forces an immediate garbage collection from generation 0 through a specified generation.
Collect(Int32, GCCollectionMode)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value.
Collect(Int32, GCCollectionMode, Boolean)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with a value specifying whether the collection should be blocking.
Collect(Int32, GCCollectionMode, Boolean, Boolean)	Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with values that specify whether the collection should be blocking and compacting.

Collect()

Forces an immediate garbage collection of all generations.

C#

```
public static void Collect ();
```

Examples

The following example demonstrates how to use the Collect method to perform a collection on all generations of memory. The code generates a number of unused objects, and then calls the Collect method to clean them from memory.

C#

```
using System;

class MyGCCollectClass
{
   private const int maxGarbage = 1000;

   static void Main()
   {
      // Put some objects in memory.
      MyGCCollectClass.MakeSomeGarbage();
      Console.WriteLine("Memory used before collection:       {0:N0}",
                        GC.GetTotalMemory(false));

      // Collect all generations of memory.
      GC.Collect();
      Console.WriteLine("Memory used after full collection:   {0:N0}",
                        GC.GetTotalMemory(true));
   }

   static void MakeSomeGarbage()
   {
      Version vt;

      // Create objects and release them to fill up memory with unused
objects.
      for(int i = 0; i < maxGarbage; i++) {
         vt = new Version();
      }
   }
}
// The output from the example resembles the following:
//       Memory used before collection:       79,392
//       Memory used after full collection:   52,640
```

Remarks

Use this method to try to reclaim all memory that is inaccessible. It performs a blocking garbage collection of all generations.

All objects, regardless of how long they have been in memory, are considered for collection; however, objects that are referenced in managed code are not collected. Use this method to force the system to try to reclaim the maximum amount of available memory.

Starting with the .NET Framework 4.5.1, you can compact the large object heap (LOH) by setting the GCSettings.LargeObjectHeapCompactionMode property to GCLargeObjectHeapCompactionMode.CompactOnce before calling the Collect method, as the following example illustrates.

C#

```
GCSettings.LargeObjectHeapCompactionMode =
GCLargeObjectHeapCompactionMode.CompactOnce;
GC.Collect();
```

See also

- Induced Collections

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collect(Int32)

Forces an immediate garbage collection from generation 0 through a specified generation.

C#

```
public static void Collect (int generation);
```

Parameters

generation	Int32

The number of the oldest generation to be garbage collected.

Exceptions

ArgumentOutOfRangeException

`generation` is not valid.

Examples

The following example demonstrates how to use the Collect method to perform a collection on individual layers of memory. The code generates a number of unused objects, and then calls the Collect method to clean them from memory.

C#

```
using System;

namespace GCCollectIntExample
{
    class MyGCCollectClass
    {
        private const long maxGarbage = 1000;

        static void Main()
        {
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Determine the maximum number of generations the system
        // garbage collector currently supports.
            Console.WriteLine("The highest generation is {0}",
GC.MaxGeneration);

            myGCCol.MakeSomeGarbage();

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));

            // Determine the best available approximation of the number
        // of bytes currently allocated in managed memory.
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of generation 0 only.
            GC.Collect(0);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));
```

```
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of all generations up to and
including 2.
            GC.Collect(2);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));
            Console.Read();
        }

        void MakeSomeGarbage()
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
        // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

Use this method to try to reclaim memory that is inaccessible. However, using this method does not guarantee that all inaccessible memory in the specified generation is reclaimed.

If object aging is implemented, the garbage collector does not collect objects with a generation number that is higher than the specified generation. If object aging is not implemented, the garbage collector considers all objects during the garbage collection.

Use the MaxGeneration property to determine the maximum valid value of the `generation` parameter.

To have the garbage collector consider all objects regardless of their generation, use the version of this method that takes no parameters. To have the garbage collector reclaim objects based on a GCCollectionMode setting, use the GC.Collect(Int32, GCCollectionMode) method overload.

See also

- Induced Collections

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collect(Int32, GCCollectionMode)

Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value.

```
C#

public static void Collect (int generation, GCCollectionMode mode);
```

Parameters

`generation` Int32

The number of the oldest generation to be garbage collected.

`mode` GCCollectionMode

An enumeration value that specifies whether the garbage collection is forced (Default or Forced) or optimized (Optimized).

Exceptions

ArgumentOutOfRangeException

`generation` is not valid.

-or-

`mode` is not one of the GCCollectionMode values.

Examples

The following example forces a garbage collection for generation 2 objects with the Optimized setting.

C#

```
using System;

class Program
{
    static void Main(string[] args)
    {
        GC.Collect(2, GCCollectionMode.Optimized);
    }
}
```

Remarks

Use the `mode` parameter to specify whether garbage collection should occur immediately or only if the time is optimal to reclaim objects. Using this method does not guarantee that all inaccessible memory in the specified generation is reclaimed.

To adjust the intrusiveness of garbage collection during critical periods in your application, set the LatencyMode property.

The garbage collector does not collect objects with a generation number higher than specified by the `generation` parameter. Use the MaxGeneration property to determine the maximum valid value of `generation`.

To have the garbage collector consider all objects regardless of their generation, use the version of this method that takes no parameters.

To have the garbage collector reclaim objects up to a specified generation of objects, use the GC.Collect(Int32) method overload. When you specify the maximum generation, all objects are collected.

See also

- Induced Collections

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collect(Int32, GCCollectionMode, Boolean)

Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with a value specifying whether the collection should be blocking.

C#

```
public static void Collect (int generation, GCCollectionMode mode, bool
blocking);
```

Parameters

`generation` Int32

The number of the oldest generation to be garbage collected.

`mode` GCCollectionMode

An enumeration value that specifies whether the garbage collection is forced (Default or Forced) or optimized (Optimized).

`blocking` Boolean

`true` to perform a blocking garbage collection; `false` to perform a background garbage collection where possible.

Exceptions

ArgumentOutOfRangeException

`generation` is not valid.

-or-

`mode` is not one of the GCCollectionMode values.

Remarks

The following table summarizes the interaction of the `mode` and `blocking` parameters:

Expand table

`mode`	`blocking` is `true`	`blocking` is `false`
Forced or Default	A blocking collection is performed as soon as possible. If a background collection is in progress and `generation` is 0 or 1, the Collect(Int32, GCCollectionMode, Boolean) method immediately triggers a blocking collection and returns when the collection is finished. If a background collection is in progress and `generation` is 2, the method waits until the background collection is finished, triggers a blocking generation 2 collection, and then returns.	A collection is performed as soon as possible. The Collect(Int32, GCCollectionMode, Boolean) method requests a background collection, but this is not guaranteed; depending on the circumstances, a blocking collection may still be performed. If a background collection is already in progress, the method returns immediately.
Optimized	A blocking collection may be performed, depending on the state of the garbage collector and the `generation` parameter. The garbage collector tries to provide optimal performance.	A collection may be performed, depending on the state of the garbage collector. The Collect(Int32, GCCollectionMode, Boolean) method requests a background collection, but this is not guaranteed; depending on the circumstances, a blocking collection may still be performed. The garbage collector tries to provide optimal performance. If a background collection is already in progress, the method returns immediately.

If a call to the Collect(Int32, GCCollectionMode, Boolean) method performs a full blocking garbage collection, you can also compact the large object heap by setting the GCSettings.LargeObjectHeapCompactionMode property to

GCLargeObjectHeapCompactionMode.CompactOnce before calling the Collect method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collect(Int32, GCCollectionMode, Boolean, Boolean)

Forces a garbage collection from generation 0 through a specified generation, at a time specified by a GCCollectionMode value, with values that specify whether the collection should be blocking and compacting.

C#

```
public static void Collect (int generation, GCCollectionMode mode, bool
blocking, bool compacting);
```

Parameters

`generation` Int32

The number of the oldest generation to be garbage collected.

`mode` GCCollectionMode

An enumeration value that specifies whether the garbage collection is forced (Default or Forced) or optimized (Optimized).

`blocking` Boolean

`true` to perform a blocking garbage collection; `false` to perform a background garbage collection where possible.

`compacting` Boolean

`true` to compact the small object heap; `false` to sweep only.

Remarks

If `blocking` is `false`, the GC decides whether to perform a background or a blocking garbage collection. If `compacting` is `true`, it performs a blocking garbage collection.

If `compacting` is `true`, the runtime compacts the small object heap (SOH). The large object heap (LOH) is not compacted unless the GCSettings.LargeObjectHeapCompactionMode property is set to GCLargeObjectHeapCompactionMode.CompactOnce. Note that this includes all blocking garbage collections, not just full blocking garbage collections.

You can call the Collect(Int32, GCCollectionMode, Boolean, Boolean) method to reduce the managed heap to the smallest size possible, as the following code fragment illustrates.

```
C#

GCSettings.LargeObjectHeapCompactionMode =
GCLargeObjectHeapCompactionMode.CompactOnce;
GC.Collect(2, GCCollectionMode.Forced, true, true);
```

Specifying `true` for the `compacting` argument guarantees a compacting, full blocking garbage collection. Setting the GCSettings.LargeObjectHeapCompactionMode property to GCLargeObjectHeapCompactionMode.CompactOnce ensures that both the LOH and SOH are compacted.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GC.CollectionCount(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the number of times garbage collection has occurred for the specified generation of objects.

```
C#
public static int CollectionCount (int generation);
```

Parameters

`generation` Int32

The generation of objects for which the garbage collection count is to be determined.

Returns

Int32

The number of times garbage collection has occurred for the specified generation since the process was started.

Exceptions

ArgumentOutOfRangeException

`generation` is less than 0.

Remarks

If you implement your own resource management, you might need to force garbage collection periodically by calling the Collect method. Because this is an expensive operation, you can improve performance by skipping the call when a garbage collection has occurred recently. Save the value returned by CollectionCount immediately after calling Collect. The next time you need to call Collect, compare the current value

returned by CollectionCount to the saved value. If the two values are equal, no collection has occurred in the interim and it is reasonable to call Collect again.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.EndNoGCRegion Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Ends the no GC region latency mode.

C#

```
public static void EndNoGCRegion ();
```

Exceptions

InvalidOperationException

The garbage collector is not in no GC region latency mode.

-or-

The no GC region latency mode was ended previously because a garbage collection was induced.

-or-

A memory allocation exceeded the amount specified in the call to the TryStartNoGCRegion(Int64) method.

Remarks

The EndNoGCRegion method throws an InvalidOperationException if the garbage collector is not in no GC region latency mode. This occurs under any of the following conditions:

- The TryStartNoGCRegion method was not called previously.
- The call to the TryStartNoGCRegion method returned `false`.
- The call to the TryStartNoGCRegion method threw an exception.

You can prevent an exception for any of these reasons by using code such as the following:

```C#
if (GCSettings.LatencyMode == GCLatencyMode.NoGCRegion)
   GC.EndNoGCRegion();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TryStartNoGCRegion(Int64)
- LatencyMode
- Latency Modes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.GetAllocatedBytesForCurrentThread Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total number of bytes allocated to the current thread since the beginning of its lifetime.

C#

```
public static long GetAllocatedBytesForCurrentThread ();
```

Returns

Int64

The total number of bytes allocated to the current thread since the beginning of its lifetime.

Remarks

The `GetAllocatedBytesForCurrentThread` method returns the total number of bytes allocated on the managed heap during the lifetime of a thread, not the total number of bytes that have survived garbage collection. The returned value also does not include any native allocations.

This method is most useful in monitoring scenarios for measuring the difference in memory allocation between time intervals or events.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.GetConfigurationVariables Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the configurations used by the garbage collector.

C#

```
public static System.Collections.Generic.IReadOnlyDictionary<string,object>
GetConfigurationVariables ();
```

Returns

IReadOnlyDictionary<String,Object>

A read-only dictionary with the configuration names and values as the keys and values of the dictionary, respectively.

Remarks

The value of these configurations don't necessarily have to be the same as the ones that are passed by the user. For example, for the `GCHeapCount` configuration, if the user supplies a value higher than the number of CPUs, the configuration that's used is that of the number of CPUs.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GC.GetGCMemoryInfo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetGCMemoryInfo()	Gets garbage collection memory information.
GetGCMemoryInfo(GCKind)	Gets garbage collection memory information.

GetGCMemoryInfo()

Gets garbage collection memory information.

C#

```
public static GCMemoryInfo GetGCMemoryInfo ();
```

Returns

GCMemoryInfo

An object that contains information about the garbage collector's memory usage.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

GetGCMemoryInfo(GCKind)

Gets garbage collection memory information.

```
C#
public static GCMemoryInfo GetGCMemoryInfo (GCKind kind);
```

Parameters

`kind` GCKind

The kind of collection for which to retrieve memory information.

Returns

GCMemoryInfo

An object that contains information about the garbage collector's memory usage.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.GetGeneration Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the current generation number of an object.

Overloads

Expand table

GetGeneration(Object)	Returns the current generation number of the specified object.
GetGeneration(WeakReference)	Returns the current generation number of the target of a specified weak reference.

GetGeneration(Object)

Returns the current generation number of the specified object.

C#

```
public static int GetGeneration (object obj);
```

Parameters

`obj` Object

The object that generation information is retrieved for.

Returns

Int32

The current generation number of `obj`, or Int32.MaxValue.

Examples

The following example demonstrates how to use the GetGeneration method to determine the age of an object. The example then performs garbage collections to clean up memory and compare the pre and post collection memory totals in the console.

C#

```
using System;

namespace GCCollectIntExample
{
    class MyGCCollectClass
    {
        private const long maxGarbage = 1000;

        static void Main()
        {
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Determine the maximum number of generations the system
        // garbage collector currently supports.
            Console.WriteLine("The highest generation is {0}",
GC.MaxGeneration);

            myGCCol.MakeSomeGarbage();

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));

            // Determine the best available approximation of the number
        // of bytes currently allocated in managed memory.
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of generation 0 only.
            GC.Collect(0);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));

            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of all generations up to and
including 2.
            GC.Collect(2);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}",
GC.GetGeneration(myGCCol));
            Console.WriteLine("Total Memory: {0}",
```

```
GC.GetTotalMemory(false));
            Console.Read();
        }

        void MakeSomeGarbage()
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
        // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

Use this method to determine the age of an object, and then use that information with the Collect method to force the garbage collector to collect objects in the same generation. For example, use this method when you have a set of objects that are created as a group and that become inaccessible at the same time.

Starting in .NET 8, this method might return MaxValue for objects allocated on non-GC heaps. For more information, see GC.GetGeneration might return Int32.MaxValue.

See also

- MaxGeneration

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetGeneration(WeakReference)

Returns the current generation number of the target of a specified weak reference.

```
C#

public static int GetGeneration (WeakReference wo);
```

Parameters

wo WeakReference

A WeakReference that refers to the target object whose generation number is to be determined.

Returns

Int32

The current generation number of the target of `wo`, or MaxValue.

Exceptions

ArgumentException

Garbage collection has already been performed on `wo`.

ArgumentNullException

The target of the weak reference has already been garbage collected.

Examples

The following example demonstrates the use of the GetGeneration method to determine the age of a weak reference object.

```
C#

using System;

namespace GCGetGenerationWeakExample
{
    public class MyGCCollectClass
    {
        private const long maxGarbage = 1000;

        static void Main()
```

```
        {
            // Create a strong reference to an object.
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Put some objects in memory.
            myGCCol.MakeSomeGarbage();

            // Get the generation of managed memory where myGCCol is
stored.
            Console.WriteLine("The object is in generation: {0}",
GC.GetGeneration(myGCCol));

            // Perform a full garbage collection.
            // Because there is a strong reference to myGCCol, it will
            // not be garbage collected.
            GC.Collect();

            // Get the generation of managed memory where myGCCol is
stored.
            Console.WriteLine("The object is in generation: {0}",
GC.GetGeneration(myGCCol));

            // Create a WeakReference to myGCCol.
            WeakReference wkref = new WeakReference(myGCCol);
            // Remove the strong reference to myGCCol.
            myGCCol = null;

            // Get the generation of managed memory where wkref is
stored.
            Console.WriteLine("The WeakReference to the object is in
generation: {0}", GC.GetGeneration(wkref));

            // Perform another full garbage collection.
            // A WeakReference will not survive a garbage collection.
            GC.Collect();

            // Try to get the generation of managed memory where wkref is
stored.
            // Because it has been collected, an exception will be
thrown.
            try
            {
                Console.WriteLine("The WeakReference to the object is in
generation: {0}", GC.GetGeneration(wkref));
                Console.Read();
            }
            catch(Exception e)
            {
                Console.WriteLine("The WeakReference to the object has
been garbage collected: '{0}'", e);
                Console.Read();
            }
        }

        void MakeSomeGarbage()
```

```
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
                // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

Starting in .NET 8, this method might return MaxValue for objects allocated on non-GC heaps. For more information, see GC.GetGeneration might return Int32.MaxValue.

See also

- MaxGeneration

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

GC.GetTotalAllocatedBytes(Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a count of the bytes allocated over the lifetime of the process. The returned value does not include any native allocations.

C#

```
public static long GetTotalAllocatedBytes (bool precise = false);
```

Parameters

`precise` Boolean

If `true`, gather a precise number; otherwise, gather an approximate count. Gathering a precise value entails a significant performance penalty.

Returns

Int64

The total number of bytes allocated over the lifetime of the process.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GC.GetTotalMemory(Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the heap size excluding fragmentation. For example if the total GC heap size is 100mb and fragmentation, ie, space taken up by free objects, takes up 40mb, this API would report 60mb. A parameter indicates whether this method can wait a short interval before returning, to allow the system to collect garbage and finalize objects.

```
C#

public static long GetTotalMemory (bool forceFullCollection);
```

Parameters

`forceFullCollection` Boolean

`true` to indicate that this method can wait for garbage collection to occur before returning; otherwise, `false`.

Returns

Int64

The heap size, in bytes, excluding fragmentation.

Examples

The following example demonstrates how to use the GetTotalMemory method to get and display the number of bytes currently allocated in managed memory.

```
C#

using System;

namespace GCCollectIntExample
{
    class MyGCCollectClass
```

```
    {
        private const long maxGarbage = 1000;

        static void Main()
        {
            MyGCCollectClass myGCCol = new MyGCCollectClass();

            // Determine the maximum number of generations the system
        // garbage collector currently supports.
            Console.WriteLine("The highest generation is {0}",
GC.MaxGeneration);

            myGCCol.MakeSomeGarbage();

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            // Determine the best available approximation of the number
        // of bytes currently allocated in managed memory.
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of generation 0 only.
            GC.Collect(0);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));

            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));

            // Perform a collection of all generations up to and including
2.
            GC.Collect(2);

            // Determine which generation myGCCol object is stored in.
            Console.WriteLine("Generation: {0}", GC.GetGeneration(myGCCol));
            Console.WriteLine("Total Memory: {0}",
GC.GetTotalMemory(false));
            Console.Read();
        }

        void MakeSomeGarbage()
        {
            Version vt;

            for(int i = 0; i < maxGarbage; i++)
            {
                // Create objects and release them to fill up memory
        // with unused objects.
                vt = new Version();
            }
        }
    }
}
```

Remarks

If the `forceFullCollection` parameter is `true`, this method waits a short interval before returning while the system collects garbage and finalizes objects. The duration of the interval is an internally specified limit determined by the number of garbage collection cycles completed and the change in the amount of memory recovered between cycles. The garbage collector does not guarantee that all inaccessible memory is collected.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.GetTotalPauseDuration Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total amount of time paused in GC since the beginning of the process.

C#

```
public static TimeSpan GetTotalPauseDuration ();
```

Returns

TimeSpan

The total amount of time paused in GC since the beginning of the process.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.KeepAlive(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

References the specified object, which makes it ineligible for garbage collection from the start of the current routine to the point where this method is called.

C#

```
public static void KeepAlive (object? obj);
```

Parameters

obj Object

The object to reference.

Examples

The following code example creates an object at the beginning of its `Main` method and does not refer to the object again until the end, when the KeepAlive method is called. The object persists for the 30-second duration of the `Main` method, despite calls to the Collect and WaitForPendingFinalizers methods.

C#

```
using System;
using System.Threading;
using System.Runtime.InteropServices;

// A simple class that exposes two static Win32 functions.
// One is a delegate type and the other is an enumerated type.
public class MyWin32
{
    // Declare the SetConsoleCtrlHandler function
    // as external and receiving a delegate.
    [DllImport("Kernel32")]
    public static extern Boolean SetConsoleCtrlHandler(HandlerRoutine
Handler,
        Boolean Add);
```

```
    // A delegate type to be used as the handler routine
    // for SetConsoleCtrlHandler.
    public delegate Boolean HandlerRoutine(CtrlTypes CtrlType);

    // An enumerated type for the control messages
    // sent to the handler routine.
    public enum CtrlTypes
    {
        CTRL_C_EVENT = 0,
        CTRL_BREAK_EVENT,
        CTRL_CLOSE_EVENT,
        CTRL_LOGOFF_EVENT = 5,
        CTRL_SHUTDOWN_EVENT
    }
}

public class MyApp
{
    // A private static handler function in the MyApp class.
    static Boolean Handler(MyWin32.CtrlTypes CtrlType)
    {
        String message = "This message should never be seen!";

        // A switch to handle the event type.
        switch(CtrlType)
        {
            case MyWin32.CtrlTypes.CTRL_C_EVENT:
                message = "A CTRL_C_EVENT was raised by the user.";
                break;
            case MyWin32.CtrlTypes.CTRL_BREAK_EVENT:
                message = "A CTRL_BREAK_EVENT was raised by the user.";
                break;
            case MyWin32.CtrlTypes.CTRL_CLOSE_EVENT:
                message = "A CTRL_CLOSE_EVENT was raised by the user.";
                break;
            case MyWin32.CtrlTypes.CTRL_LOGOFF_EVENT:
                message = "A CTRL_LOGOFF_EVENT was raised by the user.";
                break;
            case MyWin32.CtrlTypes.CTRL_SHUTDOWN_EVENT:
                message = "A CTRL_SHUTDOWN_EVENT was raised by the user.";
                break;
        }

        // Use interop to display a message for the type of event.
        Console.WriteLine(message);

        return true;
    }

    public static void Main()
    {

        // Use interop to set a console control handler.
        MyWin32.HandlerRoutine hr = new MyWin32.HandlerRoutine(Handler);
```

```
        MyWin32.SetConsoleCtrlHandler(hr, true);

        // Give the user some time to raise a few events.
        Console.WriteLine("Waiting 30 seconds for console ctrl events...");

        // The object hr is not referred to again.
        // The garbage collector can detect that the object has no
        // more managed references and might clean it up here while
        // the unmanaged SetConsoleCtrlHandler method is still using it.

        // Force a garbage collection to demonstrate how the hr
        // object will be handled.
        GC.Collect();
        GC.WaitForPendingFinalizers();
        GC.Collect();

        Thread.Sleep(30000);

        // Display a message to the console when the unmanaged method
        // has finished its work.
        Console.WriteLine("Finished!");

        // Call GC.KeepAlive(hr) at this point to maintain a reference to
hr.
        // This will prevent the garbage collector from collecting the
        // object during the execution of the SetConsoleCtrlHandler method.
        GC.KeepAlive(hr);
        Console.Read();
    }
}
```

Remarks

The purpose of the KeepAlive method is to ensure the existence of a reference to an object that is at risk of being prematurely reclaimed by the garbage collector. A common scenario where this might happen is when there are no references to the object in managed code or data, but the object is still in use in unmanaged code such as Windows APIs, unmanaged DLLs, or methods using COM.

This method references the `obj` parameter, making that object ineligible for garbage collection from the start of the routine to the point, in execution order, where this method is called. Code this method at the end, not the beginning, of the range of instructions where `obj` must be available.

The KeepAlive method performs no operation and produces no side effects other than extending the lifetime of the object passed in as a parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.RefreshMemoryLimit Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Instructs the Garbage Collector to reconfigure itself by detecting the various memory limits on the system.

C#

```
public static void RefreshMemoryLimit ();
```

Exceptions

InvalidOperationException

The hard limit is too low. This can happen if the heap hard limit that the refresh will set, either because of new AppData settings or implied by the container memory-limit changes, is lower than what's already committed.

-or-

The hard limit is invalid. This can happen, for example, with negative heap hard-limit percentages.

Remarks

In addition to actual physical memory limit and container-limit settings, these configuration settings can be overwritten:

```
    - GCHeapHardLimit
    - GCHeapHardLimitPercent
    - GCHeapHardLimitSOH
    - GCHeapHardLimitLOH
    - GCHeapHardLimitPOH
    - GCHeapHardLimitSOHPercent
    - GCHeapHardLimitLOHPercent
    - GCHeapHardLimitPOHPercent
```

Instead of updating the environment variable (which won't be read), these override these settings by using a ulong value in the AppContext.

For example, you can use `AppContext.SetData("GCHeapHardLimit", (ulong) 100 * 1024 * 1024)` to override the GCHeapHardLimit to 100M.

This API only handles configs that can be handled when the runtime is loaded. For configs that don't have any effects on 32-bit systems (like the GCHeapHardLimit* ones), this API will not handle it.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.RegisterForFullGCNotification(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that a garbage collection notification should be raised when conditions favor full garbage collection and when the collection has been completed.

C#

```
public static void RegisterForFullGCNotification (int
maxGenerationThreshold, int largeObjectHeapThreshold);
```

Parameters

`maxGenerationThreshold` Int32

A number between 1 and 99 that specifies when the notification should be raised based on the objects allocated in generation 2.

`largeObjectHeapThreshold` Int32

A number between 1 and 99 that specifies when the notification should be raised based on objects allocated in the large object heap.

Exceptions

ArgumentOutOfRangeException

`maxGenerationThreshold` or `largeObjectHeapThreshold` is not between 1 and 99.

InvalidOperationException

This member is not available when concurrent garbage collection is enabled. See the <gcConcurrent> runtime setting for information about how to disable concurrent garbage collection.

Examples

The following example shows how to register a garbage collection notification and start a thread to monitor the status of the garbage collection notification. This code example is part of a larger example provided for Garbage Collection Notifications topic.

C#

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace GCNotify
{
    class Program
    {
        // Variable for continual checking in the
        // While loop in the WaitForFullGCProc method.
        static bool checkForNotify = false;

        // Variable for suspending work
        // (such servicing allocated server requests)
        // after a notification is received and then
        // resuming allocation after inducing a garbage collection.
        static bool bAllocate = false;

        // Variable for ending the example.
        static bool finalExit = false;

        // Collection for objects that
        // simulate the server request workload.
        static List<byte[]> load = new List<byte[]>();

        public static void Main(string[] args)
        {
            try
            {
                // Register for a notification.
                GC.RegisterForFullGCNotification(10, 10);
                Console.WriteLine("Registered for GC notification.");

                checkForNotify = true;
                bAllocate = true;

                // Start a thread using WaitForFullGCProc.
                Thread thWaitForFullGC = new Thread(new
ThreadStart(WaitForFullGCProc));
                thWaitForFullGC.Start();

                // While the thread is checking for notifications in
                // WaitForFullGCProc, create objects to simulate a server
workload.
                try
                {
```

```
                    int lastCollCount = 0;
                    int newCollCount = 0;

                    while (true)
                    {
                        if (bAllocate)
                        {
                            load.Add(new byte[1000]);
                            newCollCount = GC.CollectionCount(2);
                            if (newCollCount != lastCollCount)
                            {
                                // Show collection count when it increases:
                                Console.WriteLine("Gen 2 collection count:
{0}", GC.CollectionCount(2).ToString());
                                lastCollCount = newCollCount;
                            }

                            // For ending the example (arbitrary).
                            if (newCollCount == 500)
                            {
                                finalExit = true;
                                checkForNotify = false;
                                break;
                            }
                        }
                    }
                }
                catch (OutOfMemoryException)
                {
                    Console.WriteLine("Out of memory.");
                }

                finalExit = true;
                checkForNotify = false;
                GC.CancelFullGCNotification();
            }
            catch (InvalidOperationException invalidOp)
            {

                Console.WriteLine("GC Notifications are not supported while
concurrent GC is enabled.\n"
                    + invalidOp.Message);
            }
        }

        public static void OnFullGCApproachNotify()
        {

            Console.WriteLine("Redirecting requests.");

            // Method that tells the request queuing
            // server to not direct requests to this server.
            RedirectRequests();

            // Method that provides time to
```

```
            // finish processing pending requests.
            FinishExistingRequests();

            // This is a good time to induce a GC collection
            // because the runtime will induce a full GC soon.
            // To be very careful, you can check precede with a
            // check of the GC.GCCollectionCount to make sure
            // a full GC did not already occur since last notified.
            GC.Collect();
            Console.WriteLine("Induced a collection.");
        }

        public static void OnFullGCCompleteEndNotify()
        {
            // Method that informs the request queuing server
            // that this server is ready to accept requests again.
            AcceptRequests();
            Console.WriteLine("Accepting requests again.");
        }

        public static void WaitForFullGCProc()
        {
            while (true)
            {
                // CheckForNotify is set to true and false in Main.
                while (checkForNotify)
                {
                    // Check for a notification of an approaching
collection.
                    GCNotificationStatus s = GC.WaitForFullGCApproach();
                    if (s == GCNotificationStatus.Succeeded)
                    {
                        Console.WriteLine("GC Notification raised.");
                        OnFullGCApproachNotify();
                    }
                    else if (s == GCNotificationStatus.Canceled)
                    {
                        Console.WriteLine("GC Notification cancelled.");
                        break;
                    }
                    else
                    {
                        // This can occur if a timeout period
                        // is specified for WaitForFullGCApproach(Timeout)
                        // or WaitForFullGCComplete(Timeout)
                        // and the time out period has elapsed.
                        Console.WriteLine("GC Notification not
applicable.");
                        break;
                    }

                    // Check for a notification of a completed collection.
                    GCNotificationStatus status =
GC.WaitForFullGCComplete();
                    if (status == GCNotificationStatus.Succeeded)
```

```
                    {
                        Console.WriteLine("GC Notification raised.");
                        OnFullGCCompleteEndNotify();
                    }
                    else if (status == GCNotificationStatus.Canceled)
                    {
                        Console.WriteLine("GC Notification cancelled.");
                        break;
                    }
                    else
                    {
                        // Could be a time out.
                        Console.WriteLine("GC Notification not
applicable.");
                        break;
                    }
                }

                Thread.Sleep(500);
                // FinalExit is set to true right before
                // the main thread cancelled notification.
                if (finalExit)
                {
                    break;
                }
            }
        }

        private static void RedirectRequests()
        {
            // Code that sends requests
            // to other servers.

            // Suspend work.
            bAllocate = false;
        }

        private static void FinishExistingRequests()
        {
            // Code that waits a period of time
            // for pending requests to finish.

            // Clear the simulated workload.
            load.Clear();
        }

        private static void AcceptRequests()
        {
            // Code that resumes processing
            // requests on this server.

            // Resume work.
            bAllocate = true;
        }
```

```
    }
}
```

Remarks

For each generation, the garbage collector sets a threshold for allocations into that generation. When the size of allocations exceeds this threshold, a garbage collection is triggered on that generation. For example, if generation 2's threshold is 20MB (which means that 20MB survives generation 1 collections and is promoted into generation 2), and more than 20MB has survived generation 1 and is prompted into generation 2, the next garbage collection will be attempted as a generation 2 collection. Similarly, if the large object heap's (LOH's) threshold is 20MB and your app has allocated more than 20MB of large objects, the next garbage collection will also be attempted as a generation 2 collection (since the LOH is only collected in gen2 garbage collections).

The `maxGenerationThreshold` and `largeObjectHeapThreshold` thresholds control how much in advance you are notified before a full garbage collection occurs. The greater the threshold, the more allocations that can occur between notification and the next full garbage collection.

If you have situations in which a full garbage collection by the common language runtime would adversely affect your application's performance, you can ask to be notified when the runtime is about to do a full garbage collection and circumvent that collection by inducing a collection yourself (using the Collect method) when conditions are still favorable. In addition to changing the garbage collection schedule yourself, full GC notification is useful in following the scenarios:

- You monitor for the approach of a full garbage collection and, when you are notified that one is approaching, you reduce live data size (for example, by releasing some cache entries). As a result, when the garbage collection occurs, it is able to reclaim more memory.

- You monitor for the completion of a full garbage collection so that you can collect some statistics. For example, you might want to measure the size of the heap at GC completion so that you know the size of live data. (After a full GC, the heap is at its smallest size.)

For more information about what represents a full garbage collection, see Garbage Collection Notifications.

When you register for a garbage collection notification, you can be notified when a full garbage collection is approaching and when it is completed. This pattern resembles how

the operating system monitors for low memory notifications.

Use the following guidelines for specifying the `maxGenerationThreshold` and `largeObjectHeapThreshold` parameters:

- The larger the threshold value, the more allocations will occur between the notification and the full garbage collection.

  A larger threshold value provides more opportunities for the runtime to check for an approaching collection. This increases the likelihood that you will be notified. However, you should not set the threshold too high because that results in a more allocations before the runtime induces the next collection.

  When you induce a collection yourself upon notification using a high threshold value, fewer objects are reclaimed than would be reclaimed by the runtime's next collection.

- The smaller the threshold value, the fewer the allocations between notification and the full garbage collection.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Garbage Collection Notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

GC.RegisterNoGCRegionCallback(Int64, Action) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Registers a callback to be invoked when a certain amount of memory is allocated in the no GC region.

C#

```
public static void RegisterNoGCRegionCallback (long totalSize, Action
callback);
```

Parameters

`totalSize` Int64

The total size of the no GC region.

`callback` Action

The callback to be executed when a certain amount of memory is allocated in the no GC region.

Exceptions

ArgumentOutOfRangeException

`totalSize` is less than or equal to 0.

ArgumentNullException

The `callback` argument is `null`.

InvalidOperationException

The GC is not currently under a NoGC region.

-or-

Another callback is already registered.

-or-

The `totalSize` exceeds the size of the No GC region.

-or-

The operation to withhold memory for the callback failed.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.RemoveMemoryPressure(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Informs the runtime that unmanaged memory has been released and no longer needs to be taken into account when scheduling garbage collection.

```
C#
public static void RemoveMemoryPressure (long bytesAllocated);
```

Parameters

`bytesAllocated` Int64

The amount of unmanaged memory that has been released.

Exceptions

ArgumentOutOfRangeException

`bytesAllocated` is less than or equal to 0.

-or-

On a 32-bit computer, `bytesAllocated` is larger than Int32.MaxValue.

Remarks

In determining when to schedule garbage collection, the runtime takes into account how much managed memory is allocated. If a small managed object allocates a large amount of unmanaged memory, the runtime takes into account only the managed memory, and thus underestimates the urgency of scheduling garbage collection. The AddMemoryPressure method informs the runtime of this additional pressure on system memory, and the RemoveMemoryPressure method informs the runtime that the additional pressure has been released.

The AddMemoryPressure and RemoveMemoryPressure methods improve performance only for types that exclusively depend on finalizers to release the unmanaged resources. It's not necessary to use these methods in types that follow the dispose pattern, where finalizers are used to clean up unmanaged resources only in the event that a consumer of the type forgets to call `Dispose`. For more information on object finalization and the dispose pattern, see Cleaning Up Unmanaged Resources.

In the simplest usage pattern, a managed object allocates unmanaged memory in the constructor and releases it in the `Finalize` method. Call the AddMemoryPressure method after allocating the unmanaged memory, and call the RemoveMemoryPressure method after releasing it.

In more complicated scenarios, where the unmanaged memory allocation changes substantially during the lifetime of the managed object, you can call the AddMemoryPressure and RemoveMemoryPressure methods to communicate these incremental changes to the runtime.

> **Caution**
>
> You must ensure that you remove exactly the amount of pressure you add. Failing to do so can adversely affect the performance of the system in applications that run for long periods of time.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

GC.ReRegisterForFinalize(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Requests that the system call the finalizer for the specified object for which SuppressFinalize(Object) has previously been called.

C#

```
public static void ReRegisterForFinalize (object obj);
```

Parameters

obj Object

The object that a finalizer must be called for.

Exceptions

ArgumentNullException

`obj` is `null`.

Examples

The following example demonstrates how to use the ReRegisterForFinalize method to finalize an object a second time after garbage collection.

C#

```
using System;

namespace ReRegisterForFinalizeExample
{
    class MyMainClass
    {
        static void Main()
        {
```

```
            // Create a MyFinalizeObject.
            MyFinalizeObject mfo = new MyFinalizeObject();

            // Release the reference to mfo.
            mfo = null;

            // Force a garbage collection.
            GC.Collect();

            // At this point mfo will have gone through the first Finalize.
            // There should now be a reference to mfo in the static
            // MyFinalizeObject.currentInstance field.  Setting this value
            // to null and forcing another garbage collection will now
            // cause the object to Finalize permanently.
            MyFinalizeObject.currentInstance = null;
            GC.Collect();
        }
    }

    class MyFinalizeObject
    {
        public static MyFinalizeObject currentInstance = null;
        private bool hasFinalized = false;

        ~MyFinalizeObject()
        {
            if(hasFinalized == false)
            {
                Console.WriteLine("First finalization");

                // Put this object back into a root by creating
                // a reference to it.
                MyFinalizeObject.currentInstance = this;

                // Indicate that this instance has finalized once.
                hasFinalized = true;

                // Place a reference to this object back in the
                // finalization queue.
                GC.ReRegisterForFinalize(this);
            }
            else
            {
                Console.WriteLine("Second finalization");
            }
        }
    }
}
```

Remarks

The ReRegisterForFinalize method adds the `obj` parameter to the list of objects that request finalization before the garbage collector frees the object. The `obj` parameter must be the caller of this method.

Calling the ReRegisterForFinalize method does not guarantee that the garbage collector will call an object's finalizer.

By default, all objects that implement finalizers are added to the list of objects that require finalization; however, an object might have already been finalized or might have disabled finalization by calling the SuppressFinalize method.

A finalizer can use this method to resurrect itself or an object that it references.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- SuppressFinalize(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.SuppressFinalize(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Requests that the common language runtime not call the finalizer for the specified object.

```
C#

public static void SuppressFinalize (object obj);
```

Parameters

obj Object

The object whose finalizer must not be executed.

Exceptions

ArgumentNullException

`obj` is `null`.

Examples

The following example demonstrates how to use the SuppressFinalize method in a resource class to prevent a redundant garbage collection from being called. The example uses the dispose pattern to free both managed resources (that is, objects that implement IDisposable) and unmanaged resources.

```
C#

using System;
using System.ComponentModel;
using System.Runtime.InteropServices;

public class ConsoleMonitor : IDisposable
{
   const int STD_INPUT_HANDLE = -10;
   const int STD_OUTPUT_HANDLE = -11;
```

```
   const int STD_ERROR_HANDLE = -12;

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern IntPtr GetStdHandle(int nStdHandle);

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern bool WriteConsole(IntPtr hConsoleOutput, string lpBuffer,
          uint nNumberOfCharsToWrite, out uint lpNumberOfCharsWritten,
          IntPtr lpReserved);

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern bool CloseHandle(IntPtr handle);

   private bool disposed = false;
   private IntPtr handle;
   private Component component;

   public ConsoleMonitor()
   {
      handle = GetStdHandle(STD_OUTPUT_HANDLE);
      if (handle == IntPtr.Zero)
         throw new InvalidOperationException("A console handle is not
available.");

      component = new Component();

      string output = "The ConsoleMonitor class constructor.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);
   }

   // The finalizer represents Object.Finalize override.
   ~ConsoleMonitor()
   {
      if (handle != IntPtr.Zero) {
         string output = "The ConsoleMonitor finalizer.\n";
         uint written = 0;
         WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);
      }
      else {
         Console.Error.WriteLine("Object finalization.");
      }
      Dispose(disposing: false);
   }

   public void Write()
   {
      string output = "The Write method.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);
   }
```

```
   public void Dispose()
   {
      string output = "The Dispose method.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);

      Dispose(disposing: true);
      GC.SuppressFinalize(this);
   }

   private void Dispose(bool disposing)
   {
      string output = String.Format("The Dispose({0}) method.\n",
disposing);
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);

      // Execute if resources have not already been disposed.
      if (! disposed) {
         // If the call is from Dispose, free managed resources.
         if (disposing) {
            Console.Error.WriteLine("Disposing of managed resources.");
            if (component != null)
               component.Dispose();
         }
         // Free unmanaged resources.
         output = "Disposing of unmanaged resources.";
         WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);

         if (handle != IntPtr.Zero) {
            if (! CloseHandle(handle))
               Console.Error.WriteLine("Handle cannot be closed.");
         }
      }
      disposed = true;
   }
}

public class Example
{
   public static void Main()
   {
      Console.WriteLine("ConsoleMonitor instance....");
      ConsoleMonitor monitor = new ConsoleMonitor();
      monitor.Write();
      monitor.Dispose();
   }
}
// If the monitor.Dispose method is not called, the example displays the
following output:
//       ConsoleMonitor instance....
//       The ConsoleMonitor class constructor.
```

```
//       The Write method.
//       The ConsoleMonitor finalizer.
//       The Dispose(False) method.
//       Disposing of unmanaged resources.
//
// If the monitor.Dispose method is called, the example displays the
following output:
//       ConsoleMonitor instance....
//       The ConsoleMonitor class constructor.
//       The Write method.
//       The Dispose method.
//       The Dispose(True) method.
//       Disposing of managed resources.
//       Disposing of unmanaged resources.
```

Remarks

This method sets a bit in the object header of `obj`, which the runtime checks when calling finalizers. A finalizer, which is represented by the Object.Finalize method, is used to release unmanaged resources before an object is garbage-collected. If `obj` does not have a finalizer or the GC has already signaled the finalizer thread to run the finalizer, the call to the SuppressFinalize method has no effect.

Objects that implement the IDisposable interface can call this method from the object's IDisposable.Dispose implementation to prevent the garbage collector from calling Object.Finalize on an object that does not require it. Typically, this is done to prevent the finalizer from releasing unmanaged resources that have already been freed by the IDisposable.Dispose implementation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ReRegisterForFinalize(Object)

- Finalize()
- Dispose Pattern

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.TryStartNoGCRegion Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Attempts to disallow garbage collection during the execution of a critical path.

Overloads

Expand table

TryStartNoGCRegion(Int64, Int64, Boolean)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap, and controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.
TryStartNoGCRegion(Int64, Int64)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap.
TryStartNoGCRegion(Int64, Boolean)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available, and controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.
TryStartNoGCRegion(Int64)	Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available.

Remarks

Important

You cannot nest calls to the **TryStartNoGCRegion** method, and you should only call the **EndNoGCRegion** method if the runtime is currently in no GC region latency mode. In other words, you should not call **TryStartNoGCRegion** multiple times (after the first method call, subsequent calls will not succeed), and you should not

expect calls to **EndNoGCRegion** to succeed just because the first call to **TryStartNoGCRegion** succeeded.

TryStartNoGCRegion(Int64, Int64, Boolean)

Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap, and controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.

```
C#

public static bool TryStartNoGCRegion (long totalSize, long lohSize, bool
disallowFullBlockingGC);
```

Parameters

`totalSize` Int64

The amount of memory in bytes to allocate without triggering a garbage collection. `totalSize` - `lohSize` must be less than or equal to the size of an ephemeral segment. For information on the size of an ephemeral segment, see the "Ephemeral generations and segments" section in the Fundamentals of Garbage Collection article.

`lohSize` Int64

The number of bytes in `totalSize` to use for large object heap (LOH) allocations.

`disallowFullBlockingGC` Boolean

`true` to omit a full blocking garbage collection if the garbage collector is initially unable to allocate the specified memory on the small object heap (SOH) and LOH; otherwise, `false`.

Returns

Boolean

`true` if the runtime was able to commit the required amount of memory and the garbage collector is able to enter no GC region latency mode; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

`totalSize` - `lohSize` exceeds the ephemeral segment size.

InvalidOperationException

The process is already in no GC region latency mode.

Remarks

The TryStartNoGCRegion(Int64, Int64, Boolean) method attempts to place the garbage collector in no GC region latency mode, which disallows garbage collection while an app executes a critical region of code. If the runtime is unable to initially allocate the requested amount of memory and the `disallowFullBlockingGC` argument is `false`, the garbage collector performs a full blocking garbage collection in an attempt to free additional memory; otherwise, the allocation fails, and the method returns `false`. The garbage collector enters no GC region latency mode if it is able to allocate `lohSize` for the LOH and `totalSize` - `lohSize` for the small object heap (SOH).

`lohSize` must be large enough to handle all memory allocations that occur in the critical path for the LOH, and `totalSize` - `lohSize` must be large enough to handle all memory allocations that occur in the critical path for the SOH. This includes allocations by the app, as well as allocations that the runtime makes on the app's behalf.

Setting `disallowFullBlockingGC` to `true` to prevent a full blocking garbage collection if not enough memory is initially available is most useful in load balancing scenarios: one system can call this method and report itself as ready to accept requests if it returns `true`, and have the load balancer redirect requests to other systems if it returns `false`. It can then do a full blocking garbage collection when it's not handling requests by calling the Collect(Int32, GCCollectionMode, Boolean, Boolean) method.

> **ⓘ Important**
>
> You cannot nest calls to the **TryStartNoGCRegion** method, and you should only call the **EndNoGCRegion** method if the runtime is currently in no GC region latency mode. In other words, you should not call **TryStartNoGCRegion** multiple times (after the first method call, subsequent calls will not succeed), and you

should not expect calls to **EndNoGCRegion** to succeed just because the first call to **TryStartNoGCRegion** succeeded.

You exit the no GC region latency mode by calling the EndNoGCRegion method.

See also

- EndNoGCRegion()
- Fundamentals of Garbage Collection
- Latency Modes

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TryStartNoGCRegion(Int64, Int64)

Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available for the large object heap and the small object heap.

C#

```
public static bool TryStartNoGCRegion (long totalSize, long lohSize);
```

Parameters

`totalSize` Int64

The amount of memory in bytes to allocate without triggering a garbage collection. `totalSize` - `lohSize` must be less than or equal to the size of an ephemeral segment. For information on the size of an ephemeral segment, see the "Ephemeral generations and segments" section in the Fundamentals of Garbage Collection article.

`lohSize` Int64

The number of bytes in `totalSize` to use for large object heap (LOH) allocations.

Returns

Boolean

`true` if the runtime was able to commit the required amount of memory and the garbage collector is able to enter no GC region latency mode; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

`totalSize` - `lohSize` exceeds the ephemeral segment size.

InvalidOperationException

The process is already in no GC region latency mode.

Remarks

The TryStartNoGCRegion(Int64, Int64) method attempts to place the garbage collector in no GC region latency mode, which disallows garbage collection while an app executes a critical region of code. If the runtime is unable to initially allocate the requested amount of memory, the garbage collector performs a full blocking garbage collection in an attempt to free additional memory. The garbage collector enters no GC region latency mode if it is able to allocate `lohSize` for the LOH and `totalSize` - `lohSize` for the small object heap (SOH).

`lohSize` must be large enough to handle all memory allocations that occur in the critical path for the LOH, and `totalSize` - `lohSize` must be large enough to handle all memory allocations that occur in the critical path for the SOH. This includes allocations by the app, as well as allocations that the runtime makes on the app's behalf.

> **Important**
>
> You cannot nest calls to the **TryStartNoGCRegion** method, and you should only call the **EndNoGCRegion** method if the runtime is currently in no GC region latency mode. In other words, you should not call **TryStartNoGCRegion** multiple times (after the first method call, subsequent calls will not succeed), and you

should not expect calls to **EndNoGCRegion** to succeed just because the first call to **TryStartNoGCRegion** succeeded.

You exit the no GC region latency mode by calling the EndNoGCRegion method.

See also

- EndNoGCRegion()
- Fundamentals of Garbage Collection
- Latency Modes

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TryStartNoGCRegion(Int64, Boolean)

Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available, and controls whether the garbage collector does a full blocking garbage collection if not enough memory is initially available.

C#

```
public static bool TryStartNoGCRegion (long totalSize, bool
disallowFullBlockingGC);
```

Parameters

`totalSize` Int64

The amount of memory in bytes to allocate without triggering a garbage collection. It must be less than or equal to the size of an ephemeral segment. For information on the size of an ephemeral segment, see the "Ephemeral generations and segments" section in the Fundamentals of Garbage Collection article.

`disallowFullBlockingGC` Boolean

`true` to omit a full blocking garbage collection if the garbage collector is initially unable to allocate `totalSize` bytes; otherwise, `false`.

Returns

Boolean

`true` if the runtime was able to commit the required amount of memory and the garbage collector is able to enter no GC region latency mode; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

`totalSize` exceeds the ephemeral segment size.

InvalidOperationException

The process is already in no GC region latency mode.

Remarks

The TryStartNoGCRegion(Int64, Boolean) method attempts to place the garbage collector in no GC region latency mode, which disallows garbage collection while an app executes a critical region of code. If the runtime is unable to initially allocate the requested amount of memory and the `disallowFullBlockingGC` argument is `false`, the garbage collector performs a full blocking garbage collection in an attempt to free additional memory; otherwise, the allocation fails, and the method returns `false`. The garbage collector enters no GC region latency mode if it is able to allocate the required amount of memory, which in this case is actually 2 * `totalSize` (it attempts to allocate `totalSize` for the small object heap and `totalSize` for the large object heap).

`totalSize` must be large enough to handle all memory allocations that occur in the critical path. This includes allocations by the app, as well as allocations that the runtime makes on the app's behalf.

Setting `disallowFullBlockingGC` to `true` to prevent a full blocking garbage collection if not enough memory is initially available is most useful in load balancing scenarios: one system can call this method and report itself as ready to accept requests if it returns `true`, and have the load balancer redirect requests to other systems if it returns `false`. It can then do a full blocking garbage collection when it's not

handling requests by calling the Collect(Int32, GCCollectionMode, Boolean, Boolean) method.

> ⓘ **Important**
>
> You cannot nest calls to the **TryStartNoGCRegion** method, and you should only call the **EndNoGCRegion** method if the runtime is currently in no GC region latency mode. In other words, you should not call **TryStartNoGCRegion** multiple times (after the first method call, subsequent calls will not succeed), and you should not expect calls to **EndNoGCRegion** to succeed just because the first call to **TryStartNoGCRegion** succeeded.

You exit the no GC region latency mode by calling the EndNoGCRegion method.

See also

- EndNoGCRegion()
- Fundamentals of Garbage Collection
- Latency Modes

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TryStartNoGCRegion(Int64)

Attempts to disallow garbage collection during the execution of a critical path if a specified amount of memory is available.

C#

```
public static bool TryStartNoGCRegion (long totalSize);
```

Parameters

`totalSize` Int64

The amount of memory in bytes to allocate without triggering a garbage collection. It must be less than or equal to the size of an ephemeral segment. For information on the size of an ephemeral segment, see the "Ephemeral generations and segments" section in the Fundamentals of Garbage Collection article.

Returns

Boolean

`true` if the runtime was able to commit the required amount of memory and the garbage collector is able to enter no GC region latency mode; otherwise, `false`.

Exceptions

ArgumentOutOfRangeException

`totalSize` exceeds the ephemeral segment size.

InvalidOperationException

The process is already in no GC region latency mode.

Remarks

The TryStartNoGCRegion(Int64) method attempts to place the garbage collector in no GC region latency mode, which disallows garbage collection while an app executes a critical region of code. If the runtime is unable to initially allocate the requested amount of memory, the garbage collector performs a full blocking garbage collection in an attempt to free additional memory. The garbage collector enters no GC region latency mode if it is able to allocate the required amount of memory, which in this case is actually 2 * `totalSize` bytes (it attempts to allocate `totalSize` bytes for the small object heap and `totalSize` bytes for the large object heap).

`totalSize` must be large enough to handle all memory allocations that occur in the critical path. This includes allocations by the app, as well as allocations that the runtime makes on the app's behalf.

> **Important**

> You cannot nest calls to the **TryStartNoGCRegion** method, and you should only call the **EndNoGCRegion** method if the runtime is currently in no GC region latency mode. In other words, you should not call **TryStartNoGCRegion** multiple times (after the first method call, subsequent calls will not succeed), and you should not expect calls to **EndNoGCRegion** to succeed just because the first call to **TryStartNoGCRegion** succeeded.

You exit the no GC region latency mode by calling the EndNoGCRegion method.

See also

- EndNoGCRegion()
- Fundamentals of Garbage Collection
- Latency Modes

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GC.WaitForFullGCApproach Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.

Overloads

Expand table

WaitForFullGCApproach()	Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.
WaitForFullGCApproach(Int32)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.
WaitForFullGCApproach(TimeSpan)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.

WaitForFullGCApproach()

Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.

C#

```
public static GCNotificationStatus WaitForFullGCApproach ();
```

Returns

GCNotificationStatus

The status of the registered garbage collection notification.

Examples

The following example shows how to use this method to determine whether a full, blocking garbage collection is approaching. Whenever the status of the notification is Succeeded, the user method `OnFullGCApproachNotify` is called to perform actions in response to the approaching collection. This code example is part of a larger example provided for Garbage Collection Notifications topic.

C#

```
// Check for a notification of an approaching collection.
GCNotificationStatus s = GC.WaitForFullGCApproach();
if (s == GCNotificationStatus.Succeeded)
{
    Console.WriteLine("GC Notification raised.");
    OnFullGCApproachNotify();
}
else if (s == GCNotificationStatus.Canceled)
{
    Console.WriteLine("GC Notification cancelled.");
    break;
}
else
{
    // This can occur if a timeout period
    // is specified for WaitForFullGCApproach(Timeout)
    // or WaitForFullGCComplete(Timeout)
    // and the time out period has elapsed.
    Console.WriteLine("GC Notification not applicable.");
    break;
}
```

Remarks

Use the GCNotificationStatus enumeration returned by this method to determine the status of the current garbage collection notification that was registered by using the RegisterForFullGCNotification method. You can also use the WaitForFullGCComplete method to determine whether the full garbage collection has completed.

When the enumeration returns Succeeded, you can do tasks such as preventing additional objects from being allocated and inducing a collection yourself with the Collect method. Note that the notification does not guarantee that a full garbage

collection will occur, only that conditions have reached the threshold that are favorable for a full garbage collection to occur.

This method waits indefinitely for a garbage collection notification to be obtained. If you want to specify a time-out period for the method to return if the notification cannot be obtained, use the GC.WaitForFullGCApproach(Int32) method overload. If you call this method without specifying a time-out, you can call the CancelFullGCNotification method if you are waiting longer than preferred.

You should follow this method with a call to the WaitForFullGCComplete method to make sure that you have had a full garbage collection. Calling this method alone causes indeterminate results.

See also

- Garbage Collection Notifications

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

WaitForFullGCApproach(Int32)

Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.

C#

```
public static GCNotificationStatus WaitForFullGCApproach (int
millisecondsTimeout);
```

Parameters

`millisecondsTimeout` Int32

The length of time to wait before a notification status can be obtained. Specify -1 to wait indefinitely.

Returns

GCNotificationStatus

The status of the registered garbage collection notification.

Exceptions

ArgumentOutOfRangeException

`millisecondsTimeout` must be either non-negative or less than or equal to Int32.MaxValue or -1.

Remarks

Use the GCNotificationStatus enumeration returned by this method to determine the status of the current garbage collection notification that was registered by using the RegisterForFullGCNotification method. You can also use the WaitForFullGCComplete method to determine whether the full garbage collection has completed.

Note that this method returns immediately whenever a garbage collection notification status is obtained, regardless of the value specified by `millisecondsTimeout`. If a garbage collection notification status is not obtained before `millisecondsTimeout` times out, this method returns NotApplicable.

When the enumeration returns Succeeded, you can do tasks such as preventing additional objects from being allocated and inducing a collection yourself with the Collect method. Note that the notification does not guarantee that a full garbage collection will occur, only that conditions have reached the threshold that are favorable for a full garbage collection to occur.

You can call the CancelFullGCNotification method when you cannot wait for the time-out period to elapse.

You should follow this method with a call to the WaitForFullGCComplete method to make sure that you have had a full garbage collection. Calling this method alone causes indeterminate results.

See also

- Garbage Collection Notifications

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

WaitForFullGCApproach(TimeSpan)

Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent.

C#

```
public static GCNotificationStatus WaitForFullGCApproach (TimeSpan
timeout);
```

Parameters

`timeout` TimeSpan

The timeout on waiting for a full GC approach

Returns

GCNotificationStatus

The status of a registered full GC notification

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GC.WaitForFullGCComplete Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime has completed.

Overloads

Expand table

WaitForFullGCComplete()	Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime has completed.
WaitForFullGCComplete(Int32)	Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by common language the runtime has completed.
WaitForFullGCComplete(TimeSpan)	Returns the status of a registered notification about whether a blocking garbage collection has completed. May wait indefinitely for a full collection.

WaitForFullGCComplete()

Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime has completed.

C#

```
public static GCNotificationStatus WaitForFullGCComplete ();
```

Returns

GCNotificationStatus

The status of the registered garbage collection notification.

Examples

The following example shows how to use this method to determine whether a full garbage collection has completed. Whenever the status of the notification is Succeeded, the user method `OnFullGCCompletedNotify` is called to perform actions in response to the completed collection. This code example is part of a larger example provided for Garbage Collection Notifications topic.

C#

```
// Check for a notification of a completed collection.
GCNotificationStatus status = GC.WaitForFullGCComplete();
if (status == GCNotificationStatus.Succeeded)
{
    Console.WriteLine("GC Notification raised.");
    OnFullGCCompleteEndNotify();
}
else if (status == GCNotificationStatus.Canceled)
{
    Console.WriteLine("GC Notification cancelled.");
    break;
}
else
{
    // Could be a time out.
    Console.WriteLine("GC Notification not applicable.");
    break;
}
```

Remarks

Use the GCNotificationStatus enumeration returned by this method to determine the status of the current garbage collection notification that was registered by using the RegisterForFullGCNotification method. You can also use the WaitForFullGCApproach method to determine whether a full garbage collection is imminent.

When the enumeration returns Succeeded, you can do tasks such as resuming work and obtaining a collection count with the CollectionCount property.

This method waits indefinitely for a garbage collection notification to be obtained. If you want to specify a time-out period for the method to return if the notification cannot be obtained, use the GC.WaitForFullGCApproach(Int32) method overload. If

you call this method without specifying a time-out, you can call the CancelFullGCNotification method if you are waiting longer than preferred.

This method call should be preceded with a call to the WaitForFullGCApproach method to make sure that you have had a full garbage collection. Calling this method alone can produce indeterminate results.

See also

- Garbage Collection Notifications

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

WaitForFullGCComplete(Int32)

Returns, in a specified time-out period, the status of a registered notification for determining whether a full, blocking garbage collection by common language the runtime has completed.

C#

```
public static GCNotificationStatus WaitForFullGCComplete (int
millisecondsTimeout);
```

Parameters

`millisecondsTimeout` Int32

The length of time to wait before a notification status can be obtained. Specify -1 to wait indefinitely.

Returns

GCNotificationStatus

The status of the registered garbage collection notification.

Exceptions

InvalidOperationException

`millisecondsTimeout` must be either non-negative or less than or equal to Int32.MaxValue or -1.

Remarks

Use the GCNotificationStatus enumeration returned by this method to determine the status of the current garbage collection notification that was registered by using the RegisterForFullGCNotification method. You can also use the WaitForFullGCApproach method to determine whether a full garbage collection is imminent.

Note that this method returns immediately whenever a garbage collection notification status is obtained, regardless of the value specified by `millisecondsTimeout`. If a garbage collection notification status is not obtained before `millisecondsTimeout` times out, this method returns NotApplicable.

When the enumeration returns Succeeded, you can do tasks such as resuming work and obtaining a collection count with the CollectionCount property.

You can call the CancelFullGCNotification method when you cannot wait for the time-out period to elapse.

This method call should be preceded with a call to the WaitForFullGCApproach method to make sure that you have had a full garbage collection. Calling this method alone can produce indeterminate results.

See also

- Garbage Collection Notifications

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

WaitForFullGCComplete(TimeSpan)

Returns the status of a registered notification about whether a blocking garbage collection has completed. May wait indefinitely for a full collection.

C#

```
public static GCNotificationStatus WaitForFullGCComplete (TimeSpan
timeout);
```

Parameters

`timeout` TimeSpan

The timeout on waiting for a full collection

Returns

GCNotificationStatus

The status of a registered full GC notification

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

GC.WaitForPendingFinalizers Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Suspends the current thread until the thread that is processing the queue of finalizers has emptied that queue.

C#

```
public static void WaitForPendingFinalizers ();
```

Examples

The following example demonstrates how to use the WaitForPendingFinalizers method to suspend the current thread until finalization of all the collected objects is complete.

C#

```
using System;

namespace WaitForPendingFinalizersExample
{
   class MyWaitForPendingFinalizersClass
   {
    // You can increase this number to fill up more memory.
    const int numMfos = 1000;
    // You can increase this number to cause more
    // post-finalization work to be done.
    const int maxIterations = 100;

    static void Main(string[] args)
    {
       MyFinalizeObject mfo = null;

       // Create and release a large number of objects
       // that require finalization.
       for(int j = 0; j < numMfos; j++)
       {
          mfo = new MyFinalizeObject();
       }
```

```
        //Release the last object created in the loop.
        mfo = null;

        //Force garbage collection.
        GC.Collect();

        // Wait for all finalizers to complete before continuing.
        // Without this call to GC.WaitForPendingFinalizers,
        // the worker loop below might execute at the same time
        // as the finalizers.
        // With this call, the worker loop executes only after
        // all finalizers have been called.
        GC.WaitForPendingFinalizers();

        // Worker loop to perform post-finalization code.
        for(int i = 0; i < maxIterations; i++)
        {
           Console.WriteLine("Doing some post-finalize work");
        }
     }
    }

    class MyFinalizeObject
    {
     // Make this number very large to cause the finalizer to
     // do more work.
     private const int maxIterations = 10000;

     ~MyFinalizeObject()
     {
        Console.WriteLine("Finalizing a MyFinalizeObject");

        // Do some work.
        for(int i = 0; i < maxIterations; i++)
        {
           // This method performs no operation on i, but prevents
           // the JIT compiler from optimizing away the code inside
           // the loop.
           GC.KeepAlive(i);
        }
         }
     }
 }
```

Remarks

When the garbage collector finds objects that can be reclaimed, it checks each object to determine the object's finalization requirements. If an object implements a finalizer and has not disabled finalization by calling SuppressFinalize, the object is placed in a list of objects that are marked as ready for finalization. The garbage collector calls the Finalize

methods for the objects in this list and removes the entries from the list. This method blocks until all finalizers have run to completion.

The thread on which finalizers are run is unspecified, so there is no guarantee that this method will terminate. However, this thread can be interrupted by another thread while the WaitForPendingFinalizers method is in progress. For example, you can start another thread that waits for a period of time and then interrupts this thread if this thread is still suspended.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCCollectionMode Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the behavior for a forced garbage collection.

C#

```
public enum GCCollectionMode
```

Inheritance Object → ValueType → Enum → GCCollectionMode

Fields

Expand table

Aggressive	3	Requests that the garbage collector decommit as much memory as possible.
Default	0	The default setting for this enumeration, which is currently Forced.
Forced	1	Forces the garbage collection to occur immediately.
Optimized	2	Allows the garbage collector to determine whether the current time is optimal to reclaim objects.

Examples

The following example forces a garbage collection for generation 2 objects with the Optimized setting.

C#

```
using System;

class Program
{
    static void Main(string[] args)
    {
```

```
        GC.Collect(2, GCCollectionMode.Optimized);
    }
}
```

Remarks

Use the GC.Collect method overload to specify the GCCollectionMode value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Induced Collections

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCGenerationInfo Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the size and the fragmenation of a generation on entry and on exit of the GC reported in GCMemoryInfo.

C#

```
public readonly struct GCGenerationInfo
```

Inheritance Object → ValueType → GCGenerationInfo

Properties

Expand table

FragmentationAfterBytes	Gets the fragmentation in bytes on exit from the reported collection.
FragmentationBeforeBytes	Gets the fragmentation in bytes on entry to the reported collection.
SizeAfterBytes	Gets the size in bytes on exit from the reported collection.
SizeBeforeBytes	Gets the size in bytes on entry to the reported collection.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCGenerationInfo.FragmentationAfterBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fragmentation in bytes on exit from the reported collection.

C#

```
public long FragmentationAfterBytes { get; }
```

Property Value

Int64

A number representing the fragmentation in bytes on exit from the reported collection.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCGenerationInfo.FragmentationBeforeBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fragmentation in bytes on entry to the reported collection.

C#

```
public long FragmentationBeforeBytes { get; }
```

Property Value

Int64

A number representing the fragmentation in bytes on entry to the reported collection.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCGenerationInfo.SizeAfterBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the size in bytes on exit from the reported collection.

C#

```
public long SizeAfterBytes { get; }
```

Property Value

Int64

A number representing the size in bytes on exit from the reported collection.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCGenerationInfo.SizeBeforeBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the size in bytes on entry to the reported collection.

C#

```
public long SizeBeforeBytes { get; }
```

Property Value

Int64

A number representing the size in bytes on entry to the reported collection.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCKind Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the kind of a garbage collection.

C#

```
public enum GCKind
```

Inheritance Object → ValueType → Enum → GCKind

Fields

Expand table

Any	0	Any kind of collection.
Background	3	A background collection. This is always a generation 2 collection.
Ephemeral	1	A gen0 or gen1 collection.
FullBlocking	2	A blocking gen2 collection.

Remarks

A GC can be one of the 3 kinds - ephemeral, full blocking, or background. Their frequencies are very different. Ephemeral GCs happen much more often than the other two kinds. Background GCs usually happen infrequently, and full blocking GCs usually happen very infrequently. In order to sample the very infrequent GCs, collections are separated into kinds so callers can ask for all three kinds while maintaining a reasonable sampling rate. For example, if you're sampling once every second, without this distinction, you may never observe a background GC. With this distinction, you can always get info of the last GC of the kind you specify.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCMemoryInfo Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a set of APIs that can be used to retrieve garbage collection information.

C#

```
public readonly struct GCMemoryInfo
```

Inheritance Object → ValueType → GCMemoryInfo

Remarks

A garbage collection (GC) is identified by its Index, which starts from 1 and increases with each GC. If you ask for a GC that doesn't exist, you'll get all 0's in the info, including the Index. For example, you'll get 0's if you call the GetGCMemoryInfo method before a GC has happened, or you ask for a GC of GCKind.FullBlocking and no full blocking GCs have happened. You can use index 0 to detect that no GCs, or no GCs of the kind you specified, have occurred.

Properties

Expand table

Compacted	Gets a value that indicates if this is a compacting GC or not.
Concurrent	Gets a value that indicates if this is a concurrent GC (background GC) or not.
FinalizationPendingCount	Gets the number of objects that are ready for finalization that this GC observed.
FragmentedBytes	Gets the total fragmentation when the last garbage collection occurred.
Generation	Gets the generation this GC collected. Collecting a generation also collects younger generations.

GenerationInfo	Gets the generation information for all generations.
HeapSizeBytes	Gets the total heap size when the last garbage collection occurred.
HighMemoryLoadThresholdBytes	Gets the high memory load threshold when the last garbage collection occurred.
Index	Gets the index of this GC.
MemoryLoadBytes	Gets the physical memory load when the last garbage collection occurred.
PauseDurations	Gets the durations of the pauses when all managed threads are suspended.
PauseTimePercentage	Gets the pause time percentage in the GC so far. If the pause time is 1.2%, this property's value is 1.2. The value is calculated by taking the sum of all GC pauses so far and dividing that by the total elapsed time of the process since the runtime was loaded. This running counter is updated at the end of each GC. It does not distinguish between GCKind. That is, at every GC, the calculated value is updated, and when you access this property, it gets the latest calculated value.
PinnedObjectsCount	Gets the number of pinned objects this GC observed.
PromotedBytes	Gets the promoted bytes for this GC.
TotalAvailableMemoryBytes	Gets the total available memory for the garbage collector to use when the last garbage collection occurred.
TotalCommittedBytes	Gets the total committed bytes of the managed heap.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

GCMemoryInfo.Compacted Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates if this is a compacting GC or not.

C#

```
public bool Compacted { get; }
```

Property Value

Boolean

`true` if this is a compacting GC; `false` otherwise.

Remarks

A compacting GC is when space occupied by dead objects is reclaimed and surviving objects are compacted.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

GCMemoryInfo.Concurrent Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates if this is a concurrent GC (background GC) or not.

C#

```
public bool Concurrent { get; }
```

Property Value

Boolean

`true` if this is a concurrent GC (background GC); `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

See also

- Background garbage collection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

GCMemoryInfo.FinalizationPendingCount Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of objects that are ready for finalization that this GC observed.

C#

```
public long FinalizationPendingCount { get; }
```

Property Value

Int64

The total number of objects observed by this GC that are ready for finalization.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCMemoryInfo.FragmentedBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total fragmentation when the last garbage collection occurred.

```
C#
public long FragmentedBytes { get; }
```

Property Value

Int64

The total fragmentation when the last garbage collection occurred.

Remarks

For example, the application has the following five objects:

`| OBJ_A | OBJ_B | OBJ_C | OBJ_D | OBJ_E |`

If `OBJ_B`, `OBJ_C`, and `OBJ_E` are garbage collected but the heap is not compacted, the resulting heap will look like the following:

`| OBJ_A | F | OBJ_D |`

The memory between `OBJ_A` and `OBJ_D`, which is marked `F`, is considered part of the `FragmentedBytes` and is used to allocate new objects.

The memory after `OBJ_D` is not considered part of the `FragmentedBytes` but is also used to allocate new objects.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCMemoryInfo.Generation Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the generation this GC collected. Collecting a generation also collects younger generations.

C#

```
public int Generation { get; }
```

Property Value

Int32

The generation this GC collected.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCMemoryInfo.GenerationInfo Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the generation information for all generations.

C#

```
public ReadOnlySpan<GCGenerationInfo> GenerationInfo { get; }
```

Property Value

ReadOnlySpan<GCGenerationInfo>

The generation information for all generations.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCMemoryInfo.HeapSizeBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total heap size when the last garbage collection occurred.

C#

```
public long HeapSizeBytes { get; }
```

Property Value

Int64

The total heap size, in bytes, when the last garbage collection occurred.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCMemoryInfo.HighMemoryLoadThresholdBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the high memory load threshold when the last garbage collection occurred.

C#

```
public long HighMemoryLoadThresholdBytes { get; }
```

Property Value

Int64

The high memory load threshold, in bytes, when the last garbage collection occurred.

Remarks

Most of the GC performance heuristics are based on per-process measurements. However, to avoid paging, GC is aware of the global physical memory load on the machine or VM or in the container. GC recognizes a certain memory load percentage as a "high memory load situation". When the memory load percentage is over that threshold, GC becomes more aggressive and will do more full blocking GCs if necessary to reduce the heap size.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCMemoryInfo.Index Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the index of this GC.

```
C#
public long Index { get; }
```

Property Value

Int64

The index of this GC.

Remarks

GC indices start with 1 and are increased at the beginning of a GC. Since the information is updated at the end of a GC, this means you can get the information for a background GC with a smaller index than a foreground GC that finished earlier.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

GCMemoryInfo.MemoryLoadBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the physical memory load when the last garbage collection occurred.

```
C#
public long MemoryLoadBytes { get; }
```

Property Value

Int64

The physical memory load, in bytes, when the last garbage collection occurred.

Remarks

On Windows, memory load is a field in the [MEMORYSTATUS structure](/windows/win32/api/winbase/ns-winbase-memorystatus). It's a number between 0 and 100 that specifies the approximate percentage of physical memory that's in use (0 indicates no memory use and 100 indicates full memory use). Corresponding information is given on other operating systems.

Data is only brought into physical memory on first touch. If you allocated a big object but haven't actually used it, most of its memory isn't in physical memory. In this case, the allocation won't affect the memory load significantly.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCMemoryInfo.PauseDurations Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the durations of the pauses when all managed threads are suspended.

C#

```
public ReadOnlySpan<TimeSpan> PauseDurations { get; }
```

Property Value

ReadOnlySpan<TimeSpan>

A timespan representing the pause durations.

Remarks

For blocking GCs, there is one pause. Pauses for background GC are illustrated at Background workstation vs. server GC.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

GCMemoryInfo.PauseTimePercentage Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the pause time percentage in the GC so far. If the pause time is 1.2%, this property's value is 1.2. The value is calculated by taking the sum of all GC pauses so far and dividing that by the total elapsed time of the process since the runtime was loaded. This running counter is updated at the end of each GC. It does not distinguish between GCKind. That is, at every GC, the calculated value is updated, and when you access this property, it gets the latest calculated value.

C#

```
public double PauseTimePercentage { get; }
```

Property Value

Double

The pause time percentage.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

GCMemoryInfo.PinnedObjectsCount Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of pinned objects this GC observed.

C#

```
public long PinnedObjectsCount { get; }
```

Property Value

Int64

The total number of pinned objects observed by this GC.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GCMemoryInfo.PromotedBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the promoted bytes for this GC.

```
C#
public long PromotedBytes { get; }
```

Property Value

Int64

The number of promoted bytes for this GC.

Remarks

The promoted bytes represent the objects from the generations that the garbage collection is collecting that survived.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

GCMemoryInfo.TotalAvailableMemoryBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total available memory for the garbage collector to use when the last garbage collection occurred.

```
C#
public long TotalAvailableMemoryBytes { get; }
```

Property Value

Int64

The total available memory, in bytes, for the garbage collector to use when the last garbage collection occurred.

Remarks

This property value will be the value of the `COMPlus_GCHeapHardLimit` environment variable, or the `Server.GC.HeapHardLimit` value in `runtimeconfig.json`, if either is set.

If the program is run in a container, this value is an implementation-defined fraction of the container's size.

Otherwise, the value of the property is the physical memory on the machine that was available for the garbage collector to use when the last garbage collection occurred.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GCMemoryInfo.TotalCommittedBytes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the total committed bytes of the managed heap.

```
C#
public long TotalCommittedBytes { get; }
```

Property Value

Int64

The total committed bytes of the managed heap.

Remarks

The total committed bytes is the amount of backing store for the memory. On Windows, bytes are committed when `VirtualAlloc` is called with `MEM_COMMIT`. On Linux, bytes are committed when the page mapping is changed to `PROT_WRITE | PROT_READ`.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

GCNotificationStatus Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides information about the current registration for notification of the next full garbage collection.

```
C#
public enum GCNotificationStatus
```

Inheritance Object → ValueType → Enum → GCNotificationStatus

Fields

Expand table

Canceled	2	The current registration was canceled by the user.
Failed	1	The notification failed for any reason.
NotApplicable	4	This value can result when there's no current registration for a garbage collection notification, or a full GC has happened but was done as a background GC (that is, a GC that runs mostly concurrently with user threads) instead of a full blocking GC. Background GC is enabled by default; disabling it improves the prediction accuracy but incurs larger GC pauses.
Succeeded	0	The notification was successful and the registration was not canceled.
Timeout	3	The time specified by the `millisecondsTimeout` parameter for either WaitForFullGCApproach(Int32) or WaitForFullGCComplete(Int32) has elapsed.

Examples

The following example obtains a GCNotificationStatus enumeration from the WaitForFullGCApproach method. If the enumeration returns Succeeded, it calls the custom method `OnFullGCApproachNotify` to perform actions in response to the

approaching full garbage collection. This code example is part of a larger example provided for Garbage Collection Notifications topic.

C#

```
public static void WaitForFullGCProc()
{
    while (true)
    {
        // CheckForNotify is set to true and false in Main.
        while (checkForNotify)
        {
            // Check for a notification of an approaching collection.
            GCNotificationStatus s = GC.WaitForFullGCApproach();
            if (s == GCNotificationStatus.Succeeded)
            {
                Console.WriteLine("GC Notification raised.");
                OnFullGCApproachNotify();
            }
            else if (s == GCNotificationStatus.Canceled)
            {
                Console.WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // This can occur if a timeout period
                // is specified for WaitForFullGCApproach(Timeout)
                // or WaitForFullGCComplete(Timeout)
                // and the time out period has elapsed.
                Console.WriteLine("GC Notification not applicable.");
                break;
            }

            // Check for a notification of a completed collection.
            GCNotificationStatus status = GC.WaitForFullGCComplete();
            if (status == GCNotificationStatus.Succeeded)
            {
                Console.WriteLine("GC Notification raised.");
                OnFullGCCompleteEndNotify();
            }
            else if (status == GCNotificationStatus.Canceled)
            {
                Console.WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // Could be a time out.
                Console.WriteLine("GC Notification not applicable.");
                break;
            }
        }
```

```
        Thread.Sleep(500);
        // FinalExit is set to true right before
        // the main thread cancelled notification.
        if (finalExit)
        {
            break;
        }
    }
}
```

Remarks

Use the RegisterForFullGCNotification method to register for a full garbage collection notification. Then use the WaitForFullGCApproach method or the WaitForFullGCComplete method to return a GCNotificationStatus enumeration that contains the status of the notification.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Garbage Collection Notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GenericUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser for a hierarchical URI.

```
C#
public class GenericUriParser : UriParser
```

Inheritance Object → UriParser → GenericUriParser

Remarks

If you want to create a parser based on a well-known scheme, use HttpStyleUriParser, FtpStyleUriParser, GopherStyleUriParser, LdapStyleUriParser, or NewsStyleUriParser.

When creating a customizable parser, the behavior of the parser is specified by passing a bitwise combination of the values available in the System.GenericUriParserOptions enumeration to the GenericUriParser constructor.

The existing System.Uri class has been extended to provide support for International Resource Identifiers (IRI) and Internationalized Domain Names (IDN). Current users will not see any change from the .NET Framework 2.0 behavior unless they specifically enable IRI. This ensures application compatibility with prior versions of the .NET Framework.

All parsers derived from GenericUriParser will not gain IRI and IDN support by default. The Default option does not include IRI and IDN support. Two new values are added to the System.GenericUriParserOptions enumeration to indicate if a custom parser supports IRI and IDN.

For more information on IRI and IDN support, see the Remarks section for the System.Uri class.

Constructors

Expand table

GenericUriParser(GenericUriParserOptions)	Create a customizable parser for a hierarchical URI.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GenericUriParser(GenericUriParserOptions) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Create a customizable parser for a hierarchical URI.

```
C#
public GenericUriParser (GenericUriParserOptions options);
```

Parameters

`options` GenericUriParserOptions

Specify the options for this GenericUriParser.

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

If you want to create a parser based on a well-known scheme, use HttpStyleUriParser, FtpStyleUriParser, GopherStyleUriParser, LdapStyleUriParser, or NewsStyleUriParser.

When creating a customizable parser, the behavior of the parser is specified by passing a bitwise combination of the values available in the System.GenericUriParserOptions enumeration to the GenericUriParser constructor.

The existing System.Uri class has been extended to provide support for International Resource Identifiers (IRI) and Internationalized Domain Names (IDN). Current users will not see any change from the .NET Framework 2.0 behavior unless they specifically enable IRI. This ensures application compatibility with prior versions of the .NET Framework.

All parsers derived from GenericUriParser will not gain IRI and IDN support by default. The Default option does not include IRI and IDN support. Two new values are added to the System.GenericUriParserOptions enumeration to indicate if a custom parser supports IRI and IDN.

The IriParsing type indicates the parser supports the parsing rules specified in RFC 3987 for International Resource Identifiers (IRI). Whether IRI is used is dictated by configuration values.

The Idn type indicates the parser supports Internationalized Domain Name (IDN) parsing (IDN) of host names. Whether IDN is used is dictated by configuration values.

The configuration setting for the System.Configuration.IdnElement is indirectly controlled by the System.Configuration.IriParsingElement configuration setting that controls IRI processing in the System.Uri class. IRI processing must be enabled for IDN processing to be possible. If IRI processing is disabled, then IDN processing will be set to the default setting where the .NET Framework 2.0 behavior is used for compatibility and IDN names are not used.

The Internationalized Domain Name (IDN) attribute only controls IDN processing. All other IRI processing (character normalization, for example) is performed by default.

For more information on IRI and IDN support, see the Remarks section for the System.Uri class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IdnElement
- IriParsingElement
- GenericUriParserOptions
- Uri

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

GenericUriParserOptions Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies options for a UriParser.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum GenericUriParserOptions
```

Inheritance Object → ValueType → Enum → GenericUriParserOptions

Attributes FlagsAttribute

Fields

Expand table

AllowEmptyAuthority	2	The parser allows a URI with no authority.
Default	0	The parser: requires an authority; converts backslashes into forward slashes; unescapes path dots, forward slashes, and back slashes; and removes trailing dots, empty segments, and dots-only segments.
DontCompressPath	128	The parser does not canonicalize the URI.
DontConvertPathBackslashes	64	The parser does not convert back slashes into forward slashes.
DontUnescapePathDotsAndSlashes	256	The parser does not unescape path dots, forward slashes, or back slashes.
GenericAuthority	1	The parser allows a registry-based authority.

Idn	512	The parser supports Internationalized Domain Name (IDN) parsing (IDN) of host names. Whether IDN is used is dictated by configuration values.
IriParsing	1024	The parser supports the parsing rules specified in RFC 3987 for International Resource Identifiers (IRI). Whether IRI is used is dictated by configuration values.
NoFragment	32	The scheme does not define a fragment part.
NoPort	8	The scheme does not define a port.
NoQuery	16	The scheme does not define a query part.
NoUserInfo	4	The scheme does not define a user information part.

Remarks

You can combine any of these options to configure a generic URI parser by passing the options as a parameter to the GenericUriParser.GenericUriParser(GenericUriParserOptions) constructor.

The existing System.Uri class has been extended to provide support for International Resource Identifiers (IRI) based on RFC 3987. Current users will not see any change from the .NET Framework 2.0 behavior unless they specifically enable IRI. This ensures application compatibility with prior versions of the .NET Framework.

The configuration setting for the System.Configuration.IdnElement is indirectly controlled by the System.Configuration.IriParsingElement configuration setting that controls IRI processing in the System.Uri class. IRI processing must be enabled for IDN processing to be possible. If IRI processing is disabled, then IDN processing will be set to the default setting where the .NET Framework 2.0 behavior is used for compatibility and IDN names are not used.

The Internationalized Domain Name (IDN) attribute only controls IDN processing. All other IRI processing (character normalization, for example) is performed by default.

For more information on IRI support, see the Remarks section for the System.Uri class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.0, 1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IdnElement
- IriParsingElement
- Uri

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GopherStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the Gopher scheme.

C#

```
public class GopherStyleUriParser : UriParser
```

Inheritance Object → UriParser → GopherStyleUriParser

Constructors

Expand table

GopherStyleUriParser()	Creates a customizable parser based on the Gopher scheme.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)

IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

GopherStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a customizable parser based on the Gopher scheme.

C#

```
public GopherStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a globally unique identifier (GUID).

C#

```
public readonly struct Guid : IComparable, IComparable<Guid>,
IEquatable<Guid>, IParsable<Guid>, ISpanFormattable, ISpanParsable<Guid>,
IUtf8SpanFormattable
```

Inheritance Object → ValueType → Guid

Implements IComparable , IComparable<Guid> , IEquatable<Guid> , IFormattable , ISpanFormattable , IParsable<Guid> , IParsable<TSelf> , ISpanParsable<Guid> , IUtf8SpanFormattable

Examples

The following example uses the System.Runtime.InteropServices.GuidAttribute class to assign a GUID to an interface and to a user-defined class. It retrieves the value of the GUID by calling the GetCustomAttribute method, and compares it with two other GUIDs to determine whether they are equal.

C#

```
using System;
using System.Runtime.InteropServices;

// Guid for the interface IMyInterface.
[Guid("F9168C5E-CEB2-4faa-B6BF-329BF39FA1E4")]
interface IMyInterface
{
    void MyMethod();
}

// Guid for the coclass MyTestClass.
```

```
[Guid("936DA01F-9ABD-4d9d-80C7-02AF85C822A8")]
public class MyTestClass : IMyInterface
{
    public void MyMethod() {}

    public static void Main( string []args )
    {
        GuidAttribute IMyInterfaceAttribute = (GuidAttribute)
Attribute.GetCustomAttribute(typeof(IMyInterface), typeof(GuidAttribute));

        System.Console.WriteLine("IMyInterface Attribute: " +
IMyInterfaceAttribute.Value );

        // Use the string to create a guid.
        Guid myGuid1 = new Guid(IMyInterfaceAttribute.Value );
        // Use a byte array to create a guid.
        Guid myGuid2 = new Guid(myGuid1.ToByteArray());

        if (myGuid1.Equals(myGuid2))
            System.Console.WriteLine("myGuid1 equals myGuid2");
        else
            System.Console.WriteLine("myGuid1 does not equal myGuid2" );

        // Equality operator can also be used to determine if two guids have
same value.
        if ( myGuid1 == myGuid2 )
            System.Console.WriteLine( "myGuid1 == myGuid2" );
        else
            System.Console.WriteLine( "myGuid1 != myGuid2" );
    }
}
// The example displays the following output:
//       IMyInterface Attribute: F9168C5E-CEB2-4faa-B6BF-329BF39FA1E4
//       myGuid1 equals myGuid2
//       myGuid1 == myGuid2
```

Note that the GuidAttribute attribute is typically used in an application to expose a type to COM. If you compile this example, you can run the Assembly Registration tool (Regasm.exe) on the generated assembly to create registry (.reg) and type library (.tlb) files. The .reg file can be used to register the coclass in the registry, and the .tlb file can provide metadata for COM interop.

Remarks

A GUID is a 128-bit integer (16 bytes) that can be used across all computers and networks wherever a unique identifier is required. Such an identifier has a very low probability of being duplicated.

Constructors

Expand table

Guid(Byte[])	Initializes a new instance of the Guid structure by using the specified array of bytes.
Guid(Int32, Int16, Int16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)	Initializes a new instance of the Guid structure by using the specified integers and bytes.
Guid(Int32, Int16, Int16, Byte[])	Initializes a new instance of the Guid structure by using the specified integers and byte array.
Guid(ReadOnlySpan<Byte>)	Initializes a new instance of the Guid structure by using the value represented by the specified read-only span of bytes.
Guid(ReadOnlySpan<Byte>, Boolean)	
Guid(String)	Initializes a new instance of the Guid structure by using the value represented by the specified string.
Guid(UInt32, UInt16, UInt16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)	Initializes a new instance of the Guid structure by using the specified unsigned integers and bytes.

Fields

Expand table

Empty	A read-only instance of the Guid structure whose value is all zeros.

Methods

Expand table

CompareTo(Guid)	Compares this instance to a specified Guid object and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
Equals(Guid)	Returns a value indicating whether this instance and a specified Guid object represent the same value.
Equals(Object)	Returns a value that indicates whether this instance is

	equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
NewGuid()	Initializes a new instance of the Guid structure.
Parse(ReadOnlySpan<Char>)	Converts a read-only character span that represents a GUID to the equivalent Guid structure.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String)	Converts the string representation of a GUID to the equivalent Guid structure.
Parse(String, IFormatProvider)	Parses a string into a value.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Converts the character span representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.
ParseExact(String, String)	Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.
ToByteArray()	Returns a 16-element byte array that contains the value of this instance.
ToByteArray(Boolean)	
ToString()	Returns a string representation of the value of this instance in registry format.
ToString(String)	Returns a string representation of the value of this Guid instance, according to the provided format specifier.
ToString(String, IFormatProvider)	Returns a string representation of the value of this instance of the Guid class, according to the provided format specifier and culture-specific format information.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>)	
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>)	Tries to format the current GUID instance into the provided character span.
TryParse(ReadOnlySpan<Char>, Guid)	Converts the specified read-only span of characters containing the representation of a GUID to the equivalent Guid structure.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Guid)	Tries to parse a span of characters into a value.

TryParse(String, Guid)	Converts the string representation of a GUID to the equivalent Guid structure.
TryParse(String, IFormatProvider, Guid)	Tries to parse a string into a value.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, Guid)	Converts span of characters representing the GUID to the equivalent Guid structure, provided that the string is in the specified format.
TryParseExact(String, String, Guid)	Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.
TryWriteBytes(Span<Byte>)	Tries to write the current GUID instance into a span of bytes.
TryWriteBytes(Span<Byte>, Boolean, Int32)	

Operators

Expand table

Equality(Guid, Guid)	Indicates whether the values of two specified Guid objects are equal.
GreaterThan(Guid, Guid)	Compares two values to determine which is greater.
GreaterThanOrEqual(Guid, Guid)	Compares two values to determine which is greater or equal.
Inequality(Guid, Guid)	Indicates whether the values of two specified Guid objects are not equal.
LessThan(Guid, Guid)	Compares two values to determine which is less.
LessThanOrEqual(Guid, Guid)	Compares two values to determine which is less or equal.

Explicit Interface Implementations

Expand table

ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Guid structure.

Overloads

Expand table

Guid(Byte[])	Initializes a new instance of the Guid structure by using the specified array of bytes.
Guid(ReadOnlySpan<Byte>)	Initializes a new instance of the Guid structure by using the value represented by the specified read-only span of bytes.
Guid(String)	Initializes a new instance of the Guid structure by using the value represented by the specified string.
Guid(ReadOnlySpan<Byte>, Boolean)	
Guid(Int32, Int16, Int16, Byte[])	Initializes a new instance of the Guid structure by using the specified integers and byte array.
Guid(Int32, Int16, Int16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)	Initializes a new instance of the Guid structure by using the specified integers and bytes.
Guid(UInt32, UInt16, UInt16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)	Initializes a new instance of the Guid structure by using the specified unsigned integers and bytes.

Guid(Byte[])

Initializes a new instance of the Guid structure by using the specified array of bytes.

C#

```
public Guid (byte[] b);
```

Parameters

b Byte[]

A 16-element byte array containing values with which to initialize the GUID.

Exceptions

ArgumentNullException

`b` is `null`.

ArgumentException

`b` is not 16 bytes long.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Guid(ReadOnlySpan<Byte>)

Initializes a new instance of the Guid structure by using the value represented by the specified read-only span of bytes.

C#

```
public Guid (ReadOnlySpan<byte> b);
```

Parameters

b ReadOnlySpan<Byte>

A read-only span containing the bytes representing the GUID. The span must be exactly 16 bytes long.

Exceptions

ArgumentException

The span must be exactly 16 bytes long.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Guid(String)

Initializes a new instance of the Guid structure by using the value represented by the specified string.

C#

```
public Guid (string g);
```

Parameters

g String

A string that contains a GUID in one of the following formats ("d" represents a hexadecimal digit whose case is ignored):

32 contiguous hexadecimal digits:

dddddddddddddddddddddddddddddddd

-or-

Groups of 8, 4, 4, 4, and 12 hexadecimal digits with hyphens between the groups. The entire GUID can optionally be enclosed in matching braces or parentheses:

dddddddd-dddd-dddd-dddd-dddddddddddd

-or-

{dddddddd-dddd-dddd-dddd-dddddddddddd}

-or-

(dddddddd-dddd-dddd-dddd-dddddddddddd)

-or-

Groups of 8, 4, and 4 hexadecimal digits, and a subset of eight groups of 2 hexadecimal digits, with each group prefixed by "0x" or "0X", and separated by commas. The entire GUID, as well as the subset, is enclosed in matching braces:

{0xdddddddd, 0xdddd, 0xdddd,{0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd}}

All braces, commas, and "0x" prefixes are required. All embedded spaces are ignored. All leading zeros in a group are ignored.

The hexadecimal digits shown in a group are the maximum number of meaningful hexadecimal digits that can appear in that group. You can specify from 1 to the number of hexadecimal digits shown for a group. The specified digits are assumed to be the low-order digits of the group.

Exceptions

ArgumentNullException

`g` is `null`.

FormatException

The format of `g` is invalid.

OverflowException

The format of `g` is invalid.

Examples

The following example passes each string listed in the Remarks section to the Guid(String) constructor.

C#

```
string[] guidStrings = { "ca761232ed4211cebacd00aa0057b223",
                         "CA761232-ED42-11CE-BACD-00AA0057B223",
                         "{CA761232-ED42-11CE-BACD-00AA0057B223}",
                         "(CA761232-ED42-11CE-BACD-00AA0057B223)",
                         "{0xCA761232, 0xED42, 0x11CE, {0xBA, 0xCD, 0x00,
0xAA, 0x00, 0x57, 0xB2, 0x23}}" };
foreach (var guidString in guidStrings)
```

```
{
    var guid = new Guid(guidString);
    Console.WriteLine($"Original string: {guidString}");
    Console.WriteLine($"Guid:            {guid}");
    Console.WriteLine();
}

// The example displays the following output:
//    Original string: ca761232ed4211cebacd00aa0057b223
//    Guid:            ca761232-ed42-11ce-bacd-00aa0057b223
//
//    Original string: CA761232-ED42-11CE-BACD-00AA0057B223
//    Guid:            ca761232-ed42-11ce-bacd-00aa0057b223
//
//    Original string: {CA761232-ED42-11CE-BACD-00AA0057B223}
//    Guid:            ca761232-ed42-11ce-bacd-00aa0057b223
//
//    Original string: (CA761232-ED42-11CE-BACD-00AA0057B223)
//    Guid:            ca761232-ed42-11ce-bacd-00aa0057b223
//
//    Original string: {0xCA761232, 0xED42, 0x11CE, {0xBA, 0xCD, 0x00,
0xAA, 0x00, 0x57, 0xB2, 0x23}}
//    Guid:            ca761232-ed42-11ce-bacd-00aa0057b223
```

Remarks

The alphabetic hexadecimal digits in the `g` parameter can be uppercase or lowercase. For example, the following strings represent the same GUID:

"ca761232ed4211cebacd00aa0057b223"

"CA761232-ED42-11CE-BACD-00AA0057B223"

"{CA761232-ED42-11CE-BACD-00AA0057B223}"

"(CA761232-ED42-11CE-BACD-00AA0057B223)"

"{0xCA761232, 0xED42, 0x11CE, {0xBA, 0xCD, 0x00, 0xAA, 0x00, 0x57, 0xB2, 0x23}}"

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Guid(ReadOnlySpan<Byte>, Boolean)

C#

```
public Guid (ReadOnlySpan<byte> b, bool bigEndian);
```

Parameters

b ReadOnlySpan<Byte>

bigEndian Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Guid(Int32, Int16, Int16, Byte[])

Initializes a new instance of the Guid structure by using the specified integers and byte array.

C#

```
public Guid (int a, short b, short c, byte[] d);
```

Parameters

a Int32

The first 4 bytes of the GUID.

b Int16

The next 2 bytes of the GUID.

c Int16

The next 2 bytes of the GUID.

d Byte[]

The remaining 8 bytes of the GUID.

Exceptions

ArgumentNullException

`d` is `null`.

ArgumentException

`d` is not 8 bytes long.

Examples

`Guid(1,2,3,new byte[]{0,1,2,3,4,5,6,7})` creates a Guid that corresponds to "00000001-0002-0003-0001-020304050607".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Guid(Int32, Int16, Int16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)

Initializes a new instance of the Guid structure by using the specified integers and bytes.

C#

```
public Guid (int a, short b, short c, byte d, byte e, byte f, byte g,
byte h, byte i, byte j, byte k);
```

Parameters

a Int32

The first 4 bytes of the GUID.

b Int16

The next 2 bytes of the GUID.

c Int16

The next 2 bytes of the GUID.

d Byte

The next byte of the GUID.

e Byte

The next byte of the GUID.

f Byte

The next byte of the GUID.

g Byte

The next byte of the GUID.

h Byte

The next byte of the GUID.

i Byte

The next byte of the GUID.

`j` Byte

The next byte of the GUID.

`k` Byte

The next byte of the GUID.

Examples

The following example creates a GUID whose string representation is "0000000a-000b-000c-0001-020304050607".

```C#
var g = new Guid(0xA, 0xB, 0xC,
                 new Byte[] { 0, 1, 2, 3, 4, 5, 6, 7 } );
Console.WriteLine($"{g:B}");

// The example displays the following output:
//        {0000000a-000b-000c-0001-020304050607}
```

Remarks

Specifying individual bytes in this manner can be used to circumvent byte order restrictions (big-endian or little-endian byte order) on particular types of computers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Guid(UInt32, UInt16, UInt16, Byte, Byte, Byte, Byte, Byte, Byte, Byte, Byte)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the Guid structure by using the specified unsigned integers and bytes.

C#

```
[System.CLSCompliant(false)]
public Guid (uint a, ushort b, ushort c, byte d, byte e, byte f, byte g,
byte h, byte i, byte j, byte k);
```

Parameters

a UInt32

The first 4 bytes of the GUID.

b UInt16

The next 2 bytes of the GUID.

c UInt16

The next 2 bytes of the GUID.

d Byte

The next byte of the GUID.

e Byte

The next byte of the GUID.

f Byte

The next byte of the GUID.

g Byte

The next byte of the GUID.

`h` Byte

The next byte of the GUID.

`i` Byte

The next byte of the GUID.

`j` Byte

The next byte of the GUID.

`k` Byte

The next byte of the GUID.

Attributes CLSCompliantAttribute

Remarks

Specifying the bytes in this manner avoids endianness issues.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Guid.Empty Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A read-only instance of the Guid structure whose value is all zeros.

C#

```
public static readonly Guid Empty;
```

Field Value

Guid

Remarks

You can compare a GUID with the value of the Guid.Empty field to determine whether a GUID is non-zero. The following example uses the Equality operator to compare two GUID values with Guid.Empty to determine whether they consist exclusively of zeros.

C#

```
// Create a GUID and determine whether it consists of all zeros.
Guid guid1 = Guid.NewGuid();
Console.WriteLine(guid1);
Console.WriteLine($"Empty: {guid1 == Guid.Empty}\n");

// Create a GUID with all zeros and compare it to Empty.
var bytes = new Byte[16];
var guid2 = new Guid(bytes);
Console.WriteLine(guid2);
Console.WriteLine($"Empty: {guid2 == Guid.Empty}");

// The example displays output like the following:
//       11c43ee8-b9d3-4e51-b73f-bd9dda66e29c
//       Empty: False
//
//       00000000-0000-0000-0000-000000000000
//       Empty: True
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Guid and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Guid)	Compares this instance to a specified Guid object and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(Guid)

Compares this instance to a specified Guid object and returns an indication of their relative values.

C#

```
public int CompareTo (Guid value);
```

Parameters

`value` Guid

An object to compare to this instance.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return value	Description
A negative integer	This instance is less than `value`.
Zero	This instance is equal to `value`.
A positive integer	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following example calls the CompareTo(Guid) method to compare a GUID value with two similar GUID values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Guid mainGuid = Guid.Parse("01e75c83-c6f5-4192-b57e-7427cec5560d");
      unchecked {
         Guid guid2 = new Guid(0x01e75c83, (short) 0xc6f5,
                               0x4192,
                               new Byte[] { 0xb5, 0x7e, 0x74, 0x27, 0xce,
0xc5, 0x56, 0x0c} );
         Guid guid3 = Guid.Parse("01e75c84-c6f5-4192-b57e-7427cec5560d");

         Console.WriteLine("{0} {1:F} {2}", mainGuid,
                           (Comparison) mainGuid.CompareTo(guid2),
guid2);
         Console.WriteLine("{0} {1:F} {2}", mainGuid,
                           (Comparison) mainGuid.CompareTo(guid3),
guid3);
      }
   }

   private enum Comparison
   { LessThan = -1, Equals = 0, GreaterThan = 1 }
}
```

```
// The example displays the following output:
//    01e75c83-c6f5-4192-b57e-7427cec5560d GreaterThan 01e75c83-c6f5-
4192-b57e-7427cec5560c
//    01e75c83-c6f5-4192-b57e-7427cec5560d LessThan 01e75c84-c6f5-4192-
b57e-7427cec5560d
```

Remarks

The CompareTo method compares the GUIDs as if they were values provided to the Guid(Int32, Int16, Int16, Byte[]) constructor, as follows:

- It compares the UInt32 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- It compares the first UInt16 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- It compares the second UInt16 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- If performs a byte-by-byte comparison of the next eight Byte values. When it encounters the first unequal pair, it returns the result. Otherwise, it returns 0 to indicate that the two Guid values are equal.

Note that the final eight bytes appear in the string representation of a Guid in reverse order, from low byte to high byte. For example, in the string representation of the Guid value "01e75c83-c6f5-4192-b57e-7427cec5560d", the final eight bytes are "b57e-7427cec5560d." In other words, the final eight bytes are compared on a byte-by-byte basis from left to right starting with 0xb5.

If two GUIDs have equal values for a component, the method compares the next component. When it finds a component whose values are unequal, it returns the result.

This method implements the System.IComparable<T> interface and performs slightly better than the Guid.CompareTo method because it does not have to convert the `value` parameter to a Guid value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return value	Description
A negative integer	This instance is less than `value`.
Zero	This instance is equal to `value`.
A positive integer	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Guid.

Examples

The following example uses the GuidAttribute attribute to assign a GUID to a class. It retrieves the value of this GUID by calling the Attribute.GetCustomAttribute method and passing the Value property of the returned GuidAttribute object to the Parse method. Then it compares that GUID with an array of values.

C#

```
using System;
using System.Runtime.InteropServices;

[Guid("936DA01F-9ABD-4d9d-80C7-02AF85C822A8")]
public class Example
{
   public static void Main()
   {
      GuidAttribute guidAttr = (GuidAttribute)
Attribute.GetCustomAttribute(typeof(Example),

typeof(GuidAttribute));
      Guid guidValue = Guid.Parse(guidAttr.Value);
      Object[] values = { null , 16,
                          Guid.Parse("01e75c83-c6f5-4192-b57e-
7427cec5560d"),
                          guidValue };
      foreach (var value in values) {
         try {
            Console.WriteLine("{0} and {1}: {2}", guidValue,
                              value == null ? "null" : value,
                              guidValue.CompareTo(value));
         }
         catch (ArgumentException) {
            Console.WriteLine("Cannot compare {0} and {1}", guidValue,
                              value == null ? "null" : value);
         }
      }
   }
}
// The example displays the following output:
//    936da01f-9abd-4d9d-80c7-02af85c822a8 and null: 1
//    Cannot compare 936da01f-9abd-4d9d-80c7-02af85c822a8 and 16
//    936da01f-9abd-4d9d-80c7-02af85c822a8 and 01e75c83-c6f5-4192-b57e-
```

```
7427cec5560d: 1
//    936da01f-9abd-4d9d-80c7-02af85c822a8 and 936da01f-9abd-4d9d-80c7-
02af85c822a8: 0
```

Remarks

The `value` parameter must be `null` or an instance of Guid; otherwise, an exception is thrown. Any instance of Guid, regardless of its value, is considered greater than `null`.

The CompareTo method compares the GUIDs as if they were values provided to the Guid constructor, as follows:

- It compares the Int32 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- It compares the first Int16 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- It compares the second Int16 values, and returns a result if they are unequal. If they are equal, it performs the next comparison.
- If performs a byte-by-byte comparison of the next eight Byte values. When it encounters the first unequal pair, it returns the result. Otherwise, it returns 0 to indicate that the two Guid values are equal.

If two GUIDs have equal values for a component, the method compares the next component. When it finds a component whose values are unequal, it returns the result.

Note that the final eight bytes appear in the string representation of a Guid in reverse order, from low byte to high byte. For example, in the string representation of the Guid value "01e75c83-c6f5-4192-b57e-7427cec5560d", the final eight bytes are "b57e-7427cec5560d."

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two instances of Guid represent the same value.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.
Equals(Guid)	Returns a value indicating whether this instance and a specified Guid object represent the same value.

Equals(Object)

Returns a value that indicates whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? o);
```

Parameters

o Object

The object to compare with this instance.

Returns

Boolean

`true` if `o` is a Guid that has the same value as this instance; otherwise, `false`.

Remarks

Two Guid objects are equal if they have identical byte values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Guid)

Returns a value indicating whether this instance and a specified Guid object represent the same value.

```
C#

public bool Equals (Guid g);
```

Parameters

`g` Guid

An object to compare to this instance.

Returns

Boolean

`true` if `g` is equal to this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

Two Guid objects are equal if they have identical byte values.

This method performs slightly better than the Equals method because it does not have to box the `g` parameter.

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Guid.NewGuid Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Guid structure.

C#

```
public static Guid NewGuid ();
```

Returns

Guid

A new GUID object.

Examples

The following code example creates and displays the values of two Guid objects.

C#

```
// Create and display the value of two GUIDs.
Guid g = Guid.NewGuid();
Console.WriteLine(g);
Console.WriteLine(Guid.NewGuid());

// This code example produces a result similar to the following:

// 0f8fad5b-d9cb-469f-a165-70867728950e
// 7c9e6679-7425-40de-944b-e07fc1f90ae7
```

Remarks

This is a convenient `static` method that you can call to get a new Guid. The method creates a Version 4 Universally Unique Identifier (UUID) as described in RFC 4122, Sec. 4.4. The returned Guid is guaranteed to not equal Guid.Empty.

On Windows, this function wraps a call to the CoCreateGuid function. The generated GUID contains 122 bits of strong entropy.

On non-Windows platforms, starting with .NET 6, this function calls the OS's underlying cryptographically secure pseudo-random number generator (CSPRNG) to generate 122 bits of strong entropy. In previous versions of .NET, the entropy is not guaranteed to be generated by a CSPRNG.

It is recommended that applications **not** use the *NewGuid* method for cryptographic purposes. First, since a Version 4 UUID has a partially predictable bit pattern, the *NewGuid* function cannot serve as a proper cryptographic pseudo-random function (PRF). If the output of *NewGuid* is given to a cryptographic component which requires its input to be generated by a proper PRF, the cryptographic component may not be able to maintain its security properties. Second, *NewGuid* utilizes at most 122 bits of entropy, regardless of platform. Some cryptographic components set a minimum entropy level on their inputs as a matter of policy. Such policies often set the minimum entropy level at 128 bits or higher. Passing the output of *NewGuid* to such a routine may violate its policy.

If an application requires random data for cryptographic purposes, consider using a static method on the RandomNumberGenerator class. That class provides a random number generator suitable for cryptographic use.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Guid.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>)	Converts a read-only character span that represents a GUID to the equivalent Guid structure.
Parse(String)	Converts the string representation of a GUID to the equivalent Guid structure.

Parse(String, IFormatProvider)

Parses a string into a value.

C#

```
public static Guid Parse (string s, IFormatProvider? provider);
```

Parameters

s String

The string to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Guid

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#

public static Guid Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Guid

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>)

Converts a read-only character span that represents a GUID to the equivalent Guid structure.

C#

```
public static Guid Parse (ReadOnlySpan<char> input);
```

Parameters

`input` ReadOnlySpan<Char>

A read-only span containing the bytes representing a GUID.

Returns

Guid

A structure that contains the value that was parsed.

Exceptions

FormatException

`input` is not in a recognized format.

-or-

After trimming, the length of the read-only character span is 0.

Remarks

The Parse method trims any leading or trailing white space characters from `input` and converts the remaining characters in `input` to a Guid value. This method can convert a character span that represents any of the five formats produced by the ToString methods, as shown in the following table.

Expand table

Specifier	Description	Format
`N`	32 hexadecimal digits	00000000000000000000000000000000
`D`	32 hexadecimal digits separated by hyphens	00000000-0000-0000-0000-000000000000
`B`	32 hexadecimal digits separated by hyphens, enclosed in braces	{00000000-0000-0000-0000-000000000000}
`P`	32 hexadecimal digits separated by hyphens, enclosed in parentheses	(00000000-0000-0000-0000-000000000000)
`X`	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces	{0x00000000,0x0000,0x0000, {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

The method throws a FormatException if it is unable to successfully parse the string. Some of the reasons why this might occur include:

- `input` contains characters that are not part of the hexadecimal character set.
- `input` has too many or too few characters.
- `input` is not in one of the formats recognized by the ToString method and listed in the previous table.

Use the TryParse method to catch any unsuccessful parse operations without having to handle an exception.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Parse(String)

Converts the string representation of a GUID to the equivalent Guid structure.

```
C#

public static Guid Parse (string input);
```

Parameters

`input` String

The string to convert.

Returns

Guid

A structure that contains the value that was parsed.

Exceptions

ArgumentNullException

`input` is `null`.

FormatException

`input` is not in a recognized format.

Examples

The following example creates a new GUID, converts it to three separate string representations by calling the ToString(String) method with the "B", "D", and "X" format specifiers, and then calls the Parse method to convert the strings back to Guid values.

```
C#

var originalGuid = Guid.NewGuid();
// Create an array of string representations of the GUID.
string[] stringGuids = { originalGuid.ToString("B"),
```

```
                            originalGuid.ToString("D"),
                            originalGuid.ToString("X") };

// Parse each string representation.
foreach (var stringGuid in stringGuids)
{
    try
    {
        Guid newGuid = Guid.Parse(stringGuid);
        Console.WriteLine($"Converted {stringGuid} to a Guid");
    }
    catch (ArgumentNullException)
    {
        Console.WriteLine("The string to be parsed is null.");
    }
    catch (FormatException)
    {
        Console.WriteLine($"Bad format: {stringGuid}");
    }
}

// The example displays output similar to the following:
//
//    Converted {81a130d2-502f-4cf1-a376-63edeb000e9f} to a Guid
//    Converted 81a130d2-502f-4cf1-a376-63edeb000e9f to a Guid
//    Converted {0x81a130d2,0x502f,0x4cf1,
{0xa3,0x76,0x63,0xed,0xeb,0x00,0x0e,0x9f}} to a Guid
```

Remarks

The Parse method trims any leading or trailing white space from `input` and converts the string representation of a GUID to a Guid value. This method can convert strings in any of the five formats produced by the ToString(String) and ToString(String, IFormatProvider) methods, as shown in the following table.

Expand table

Specifier	Description	Format
N	32 hexadecimal digits	00000000000000000000000000000000
D	32 hexadecimal digits separated by hyphens	00000000-0000-0000-0000-000000000000
B	32 hexadecimal digits separated by hyphens, enclosed in braces	{00000000-0000-0000-0000-000000000000}
P	32 hexadecimal digits separated by hyphens, enclosed in	(00000000-0000-0000-0000-000000000000)

Specifier	Description	Format
	parentheses	
`X`	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces	{0x00000000,0x0000,0x0000, {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

The method throws a FormatException if it is unable to successfully parse the string. Some of the reasons why this might occur include:

- `input` contains characters that are not part of the hexadecimal character set.
- `input` has too many or too few characters.
- `input` is not in one of the formats recognized by the ToString method and listed in the previous table.

Use the TryParse method to catch any unsuccessful parse operations without having to handle an exception.

See also

- TryParse(String, Guid)
- ToString()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Guid.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Converts the character span representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.
ParseExact(String, String)	Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Converts the character span representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.

C#

```
public static Guid ParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format);
```

Parameters

input ReadOnlySpan<Char>

A read-only span containing the characters representing the GUID to convert.

format ReadOnlySpan<Char>

A read-only span of characters representing one of the following specifiers that indicates the exact format to use when interpreting `input`: "N", "D", "B", "P", or "X".

Returns

Guid

A structure that contains the value that was parsed.

Remarks

The ParseExact method requires the read-only character span to convert to be exactly in the format specified by the `format` parameter, after leading and trailing white-space characters are removed. The following table shows the accepted format specifiers for the `format` parameter. "0" represents a digit; hyphens ("-"), braces ("{", "}"), and parentheses ("(", ")") appear as shown.

Expand table

Specifier	Format of the `input` parameter
N	32 hexadecimal digits: 00000000000000000000000000000000
D	32 hexadecimal digits separated by hyphens: 00000000-0000-0000-0000-000000000000
B	32 hexadecimal digits separated by hyphens, enclosed in braces: {00000000-0000-0000-0000-000000000000}
P	32 hexadecimal digits separated by hyphens, enclosed in parentheses: (00000000-0000-0000-0000-000000000000)
X	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces: {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

ParseExact(String, String)

Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.

```
C#
public static Guid ParseExact (string input, string format);
```

Parameters

`input` String

The GUID to convert.

`format` String

One of the following specifiers that indicates the exact format to use when interpreting `input`: "N", "D", "B", "P", or "X".

Returns

Guid

A structure that contains the value that was parsed.

Exceptions

ArgumentNullException

`input` or `format` is `null`.

FormatException

`input` is not in the format specified by `format`.

Examples

The following example calls the ToString(String) method with each of the supported format specifiers to generate an array of strings that represent a single GUID. These

are then passed to the ParseExact method, which successfully parses only the string that conforms to the "B" format specifier.

C#

```
// Define an array of all format specifiers.
string[] formats = { "N", "D", "B", "P", "X" };
Guid guid = Guid.NewGuid();
// Create an array of valid Guid string representations.
var stringGuids = new string[formats.Length];
for (int ctr = 0; ctr < formats.Length; ctr++)
    stringGuids[ctr] = guid.ToString(formats[ctr]);

// Parse the strings in the array using the "B" format specifier.
foreach (var stringGuid in stringGuids)
{
    try
    {
        Guid newGuid = Guid.ParseExact(stringGuid, "B");
        Console.WriteLine($"Successfully parsed {stringGuid}");
    }
    catch (ArgumentNullException)
    {
        Console.WriteLine("The string to be parsed is null.");
    }
    catch (FormatException)
    {
        Console.WriteLine($"Bad Format: {stringGuid}");
    }
}

// The example displays output similar to the following:
//
//    Bad Format: eb5c8c7d187a44e68afb81e854c39457
//    Bad Format: eb5c8c7d-187a-44e6-8afb-81e854c39457
//    Successfully parsed {eb5c8c7d-187a-44e6-8afb-81e854c39457}
//    Bad Format: (eb5c8c7d-187a-44e6-8afb-81e854c39457)
//    Bad Format: {0xeb5c8c7d,0x187a,0x44e6,
{0x8a,0xfb,0x81,0xe8,0x54,0xc3,0x94,0x57}}
```

Remarks

The ParseExact method requires the string to convert to be exactly in the format specified by the `format` parameter, after leading and trailing white-space characters are removed. The following table shows the accepted format specifiers for the `format` parameter. "0" represents a digit; hyphens ("-"), braces ("{", "}"), and parentheses ("(", ")") appear as shown.

Expand table

Specifier	Format of the `input` parameter
N	32 hexadecimal digits: 00000000000000000000000000000000
D	32 hexadecimal digits separated by hyphens: 00000000-0000-0000-0000-000000000000
B	32 hexadecimal digits separated by hyphens, enclosed in braces: {00000000-0000-0000-0000-000000000000}
P	32 hexadecimal digits separated by hyphens, enclosed in parentheses: (00000000-0000-0000-0000-000000000000)
X	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces: {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

See also

- TryParseExact(String, String, Guid)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Guid.ToByteArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToByteArray()	Returns a 16-element byte array that contains the value of this instance.
ToByteArray(Boolean)	

ToByteArray()

Returns a 16-element byte array that contains the value of this instance.

C#

```
public byte[] ToByteArray ();
```

Returns

Byte[]

A 16-element byte array.

Examples

The following example calls the NewGuid method to create a Guid value, and then calls the ToByteArray method to represent the Guid value as a byte array. It then displays both values to the console. Finally, it instantiates a new Guid value from the byte array and calls its Equals(Guid) method to show that the two Guid values are identical.

C#

```
Guid guid = Guid.NewGuid();
Console.WriteLine($"Guid: {guid}");
var bytes = guid.ToByteArray();
foreach (var byt in bytes)
    Console.Write($"{byt:X2} ");

Console.WriteLine();
var guid2 = new Guid(bytes);
Console.WriteLine($"Guid: {guid2} (Same as First Guid:
{guid2.Equals(guid)})");

// The example displays output similar to the following:
//
//    Guid: 35918bc9-196d-40ea-9779-889d79b753f0
//    C9 8B 91 35 6D 19 EA 40 97 79 88 9D 79 B7 53 F0
//    Guid: 35918bc9-196d-40ea-9779-889d79b753f0 (Same as First Guid:
True)
```

Remarks

You can use the byte array returned by this method to round-trip a Guid value by calling the Guid(Byte[]) constructor.

Note that the order of bytes in the returned byte array is different from the string representation of a Guid value. The order of the beginning four-byte group and the next two two-byte groups is reversed, whereas the order of the last two-byte group and the closing six-byte group is the same. The example provides an illustration.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToByteArray(Boolean)

```
C#
public byte[] ToByteArray (bool bigEndian);
```

Parameters

`bigEndian` Boolean

Returns

Byte[]

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string representation of the value of this instance of the Guid structure.

Overloads

Expand table

ToString(String, IFormatProvider)	Returns a string representation of the value of this instance of the Guid class, according to the provided format specifier and culture-specific format information.
ToString()	Returns a string representation of the value of this instance in registry format.
ToString(String)	Returns a string representation of the value of this Guid instance, according to the provided format specifier.

ToString(String, IFormatProvider)

Returns a string representation of the value of this instance of the Guid class, according to the provided format specifier and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

format String

A single format specifier that indicates how to format the value of this Guid. The `format` parameter can be "N", "D", "B", "P", or "X". If `format` is `null` or an empty string (""), "D" is used.

`provider` IFormatProvider

(Reserved) An object that supplies culture-specific formatting information.

Returns

String

The value of this Guid, represented as a series of lowercase hexadecimal digits in the specified format.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The value of `format` is not `null`, an empty string (""), "N", "D", "B", "P", or "X".

Remarks

The `provider` parameter is reserved for future use and does not contribute to the execution of this method. You can pass `null` in the method call.

The following table shows the accepted format specifiers for the `format` parameter. "0" represents a digit; hyphens ("-"), braces ("{", "}"), and parentheses ("(", ")") appear as shown.

Expand table

Specifier	Format of return value
`N`	32 digits: 00000000000000000000000000000000
`D`	32 digits separated by hyphens: 00000000-0000-0000-0000-000000000000
`B`	32 digits separated by hyphens, enclosed in braces: {00000000-0000-0000-0000-000000000000}

Specifier	Format of return value
`P`	32 digits separated by hyphens, enclosed in parentheses: (00000000-0000-0000-0000-000000000000)
`X`	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces: {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

The hexadecimal digits a through f are lowercase in the returned string. To convert them to uppercase, call the String.ToUpper method on the returned string.

Because the `provider` parameter is ignored, you cannot use it to provide a custom formatting solution. To represent a Guid value as a string in a format that isn't supported by the standard GUID format strings, call the String.Format(IFormatProvider, String, Object[]) method with a `provider` object that implements both the ICustomFormatter and IFormatProvider interfaces. For more information, see the "Custom Formatting with ICustomFormatter" section in the Formatting Types article.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToString()

Returns a string representation of the value of this instance in registry format.

C#

```
public override string ToString ();
```

Returns

String

The value of this Guid, formatted by using the "D" format specifier as follows:

`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

where the value of the GUID is represented as a series of lowercase hexadecimal digits in groups of 8, 4, 4, 4, and 12 digits and separated by hyphens. An example of a return value is "382c74c3-721d-4f34-80e5-57657b6cbc27". To convert the hexadecimal digits from a through f to uppercase, call the ToUpper() method on the returned string.

Remarks

This method provides a default GUID format that is sufficient for typical use; however, other versions of this method that take a `format` parameter provide a few common format variations.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Returns a string representation of the value of this Guid instance, according to the provided format specifier.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A single format specifier that indicates how to format the value of this Guid. The `format` parameter can be "N", "D", "B", "P", or "X". If `format` is `null` or an empty string (""), "D" is used.

Returns

String

The value of this Guid, represented as a series of lowercase hexadecimal digits in the specified format.

Exceptions

FormatException

The value of `format` is not `null`, an empty string (""), "N", "D", "B", "P", or "X".

Remarks

The following table shows the accepted format specifiers for the `format` parameter. "0" represents a digit; hyphens ("-"), braces ("{", "}"), and parentheses ("(", ")") appear as shown.

Expand table

Specifier	Format of return value
`N`	32 digits: 00000000000000000000000000000000
`D`	32 digits separated by hyphens: 00000000-0000-0000-0000-000000000000
`B`	32 digits separated by hyphens, enclosed in braces: {00000000-0000-0000-0000-000000000000}
`P`	32 digits separated by hyphens, enclosed in parentheses: (00000000-0000-0000-0000-000000000000)

Specifier	Format of return value
`X`	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces: {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

The hexadecimal digits a through f are lowercase in the returned string. To convert them to uppercase, call the String.ToUpper method on the returned string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>)	
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>)	Tries to format the current GUID instance into the provided character span.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>)

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default);
```

Parameters

`utf8Destination` Span<Byte>

`bytesWritten` Int32

`format` ReadOnlySpan<Char>

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>)

Tries to format the current GUID instance into the provided character span.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default);
```

Parameters

`destination` Span<Char>

The span in which to write the GUID as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters written into the span.

`format` ReadOnlySpan<Char>

A read-only span containing the character representing one of the following specifiers that indicates the exact format to use when interpreting `input`: "N", "D", "B", "P", or "X".

Returns

Boolean

`true` if the formatting operation was successful; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, IFormatProvider, Guid)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Guid)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, Guid)	Converts the specified read-only span of characters containing the representation of a GUID to the equivalent Guid structure.
TryParse(String, Guid)	Converts the string representation of a GUID to the equivalent Guid structure.

TryParse(ReadOnlySpan<Char>, IFormatProvider, Guid)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out Guid result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Guid

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Guid)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
Guid result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Guid

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, Guid)

Converts the specified read-only span of characters containing the representation of a GUID to the equivalent Guid structure.

C#

```
public static bool TryParse (ReadOnlySpan<char> input, out Guid result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters representing the GUID to convert.

`result` Guid

When this method returns, contains the parsed value. If the method returns `true`, `result` contains a valid Guid. If the method returns `false`, `result` equals Empty.

Returns

Boolean

`true` if the parse operation was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Guid)

Converts the string representation of a GUID to the equivalent Guid structure.

```
C#

public static bool TryParse (string? input, out Guid result);
```

Parameters

`input` String

A string containing the GUID to convert.

`result` Guid

When this method returns, contains the parsed value. If the method returns `true`, `result` contains a valid Guid. If the method returns `false`, `result` equals Empty.

Returns

Boolean

`true` if the parse operation was successful; otherwise, `false`.

Examples

The following example creates a new GUID, converts it to three separate string representations by calling the ToString(String) method with the "B", "D", and "X" format specifiers, and then calls the TryParse method to convert the strings back to Guid values.

```
C#

Guid originalGuid = Guid.NewGuid();
// Create an array of string representations of the GUID.
string[] stringGuids = { originalGuid.ToString("B"),
                         originalGuid.ToString("D"),
                         originalGuid.ToString("X") };
```

```
// Parse each string representation.
foreach (var stringGuid in stringGuids)
{
    if (Guid.TryParse(stringGuid, out var newGuid))
        Console.WriteLine($"Converted {stringGuid} to a Guid");
    else
        Console.WriteLine($"Unable to convert {stringGuid} to a Guid");
}

// The example displays output similar to the following:
//
//    Converted {81a130d2-502f-4cf1-a376-63edeb000e9f} to a Guid
//    Converted 81a130d2-502f-4cf1-a376-63edeb000e9f to a Guid
//    Converted {0x81a130d2,0x502f,0x4cf1,
{0xa3,0x76,0x63,0xed,0xeb,0x00,0x0e,0x9f}} to a Guid
```

Remarks

This method is like the Parse method, except that instead of returning the parsed GUID, it returns `false` if `input` is `null` or not in a recognized format, and doesn't throw an exception. It trims any leading or trailing white space from `input` and converts strings in any of the five formats recognized by the ToString(String) and ToString(String, IFormatProvider) methods, as shown in the following table.

Expand table

Specifier	Description	Format
`N`	32 digits	00000000000000000000000000000000
`D`	32 digits separated by hyphens	00000000-0000-0000-0000-000000000000
`B`	32 digits separated by hyphens, enclosed in braces	{00000000-0000-0000-0000-000000000000}
`P`	32 digits separated by hyphens, enclosed in parentheses	(00000000-0000-0000-0000-000000000000)
`X`	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces	{0x00000000,0x0000,0x0000, {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

See also

- Parse(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, Guid)	Converts span of characters representing the GUID to the equivalent Guid structure, provided that the string is in the specified format.
TryParseExact(String, String, Guid)	Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, Guid)

Converts span of characters representing the GUID to the equivalent Guid structure, provided that the string is in the specified format.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, out Guid result);
```

Parameters

`input` ReadOnlySpan<Char>

A read-only span containing the characters representing the GUID to convert.

`format` ReadOnlySpan<Char>

A read-only span containing a character representing one of the following specifiers that indicates the exact format to use when interpreting `input`: "N", "D", "B", "P", or

"X".

`result` Guid

When this method returns, contains the parsed value. If the method returns `true`, `result` contains a valid Guid. If the method returns `false`, `result` equals Empty.

Returns

Boolean

`true` if the parse operation was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String, Guid)

Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.

```
C#

public static bool TryParseExact (string? input, string? format, out Guid
result);
```

Parameters

`input` String

The GUID to convert.

`format` String

One of the following specifiers that indicates the exact format to use when interpreting `input`: "N", "D", "B", "P", or "X".

`result` Guid

When this method returns, contains the parsed value. If the method returns `true`, `result` contains a valid Guid. If the method returns `false`, `result` equals Empty.

Returns

Boolean

`true` if the parse operation was successful; otherwise, `false`.

Examples

The following example calls the ToString(String) method with each of the supported format specifiers to generate an array of strings that represent a single GUID. These are then passed to the TryParseExact method, which successfully parses the string that conforms to the "B" format specifier.

C#

```
// Define an array of all format specifiers.
string[] formats = { "N", "D", "B", "P", "X" };
Guid guid = Guid.NewGuid();
// Create an array of valid Guid string representations.
var stringGuids = new string[formats.Length];
for (int ctr = 0; ctr < formats.Length; ctr++)
    stringGuids[ctr] = guid.ToString(formats[ctr]);

// Parse the strings in the array using the "B" format specifier.
foreach (var stringGuid in stringGuids)
{
    if (Guid.TryParseExact(stringGuid, "B", out var newGuid))
        Console.WriteLine($"Successfully parsed {stringGuid}");
    else
        Console.WriteLine($"Unable to parse '{stringGuid}'");
}

// The example displays output similar to the following:
//
//    Unable to parse 'c0fb150f6bf344df984a3a0611ae5e4a'
//    Unable to parse 'c0fb150f-6bf3-44df-984a-3a0611ae5e4a'
//    Successfully parsed {c0fb150f-6bf3-44df-984a-3a0611ae5e4a}
//    Unable to parse '(c0fb150f-6bf3-44df-984a-3a0611ae5e4a)'
//    Unable to parse '{0xc0fb150f,0x6bf3,0x44df,
{0x98,0x4a,0x3a,0x06,0x11,0xae,0x5e,0x4a}}'
```

Remarks

This method requires the string to convert to be exactly in the format specified by the `format` parameter, after leading and trailing white-space characters are removed. It returns `false` if `input` is `null` or is not in the format specified by `format`, and doesn't throw an exception.

The following table shows the accepted format specifiers for the `format` parameter. "0" represents a digit; hyphens ("-"), braces ("{", "}"), and parentheses ("(", ")") appear as shown.

Expand table

Specifier	Format of the `input` parameter
N	32 digits: 00000000000000000000000000000000
D	32 digits separated by hyphens: 00000000-0000-0000-0000-000000000000
B	32 digits separated by hyphens, enclosed in braces: {00000000-0000-0000-0000-000000000000}
P	32 digits separated by hyphens, enclosed in parentheses: (00000000-0000-0000-0000-000000000000)
X	Four hexadecimal values enclosed in braces, where the fourth value is a subset of eight hexadecimal values that is also enclosed in braces: {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}

See also

- ParseExact(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.TryWriteBytes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryWriteBytes(Span<Byte>)	Tries to write the current GUID instance into a span of bytes.
TryWriteBytes(Span<Byte>, Boolean, Int32)	

TryWriteBytes(Span<Byte>)

Tries to write the current GUID instance into a span of bytes.

C#

```
public bool TryWriteBytes (Span<byte> destination);
```

Parameters

`destination` Span<Byte>

When this method returns, the GUID as a span of bytes.

Returns

Boolean

`true` if the GUID is successfully written to the specified span; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryWriteBytes(Span<Byte>, Boolean, Int32)

C#

```
public bool TryWriteBytes (Span<byte> destination, bool bigEndian, out
int bytesWritten);
```

Parameters

`destination` Span<Byte>

`bigEndian` Boolean

`bytesWritten` Int32

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Guid.Equality(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the values of two specified Guid objects are equal.

C#

```
public static bool operator == (Guid a, Guid b);
```

Parameters

a Guid

The first object to compare.

b Guid

The second object to compare.

Returns

Boolean

`true` if `a` and `b` are equal; otherwise, `false`.

Examples

The following example uses the Equality operator to compare two GUID values with Guid.Empty to determine whether they consist exclusively of zeros.

C#

```
// Create a GUID and determine whether it consists of all zeros.
Guid guid1 = Guid.NewGuid();
Console.WriteLine(guid1);
Console.WriteLine($"Empty: {guid1 == Guid.Empty}\n");

// Create a GUID with all zeros and compare it to Empty.
var bytes = new Byte[16];
```

```
var guid2 = new Guid(bytes);
Console.WriteLine(guid2);
Console.WriteLine($"Empty: {guid2 == Guid.Empty}");

// The example displays output like the following:
//       11c43ee8-b9d3-4e51-b73f-bd9dda66e29c
//       Empty: False
//
//       00000000-0000-0000-0000-000000000000
//       Empty: True
```

Remarks

The equivalent method for this operator is Guid.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.GreaterThan(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#
public static bool operator > (Guid left, Guid right);
```

Parameters

`left` Guid

The value to compare with `right`.

`right` Guid

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Guid.GreaterThanOrEqual(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

C#

```
public static bool operator >= (Guid left, Guid right);
```

Parameters

`left` Guid

The value to compare with `right`.

`right` Guid

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Guid.Inequality(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the values of two specified Guid objects are not equal.

```
C#
public static bool operator != (Guid a, Guid b);
```

Parameters

a Guid

The first object to compare.

b Guid

The second object to compare.

Returns

Boolean

`true` if `a` and `b` are not equal; otherwise, `false`.

Remarks

The equivalent method for this operator is Guid.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.LessThan(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#
public static bool operator < (Guid left, Guid right);
```

Parameters

`left` Guid

The value to compare with `right`.

`right` Guid

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Guid.LessThanOrEqual(Guid, Guid) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

C#

```
public static bool operator <= (Guid left, Guid right);
```

Parameters

`left` Guid

The value to compare with `right`.

`right` Guid

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Guid.ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance into the provided span of characters.

C#

```
bool ISpanFormattable.TryFormat (Span<char> destination, out int
charsWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Guid.IUtf8SpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
bool IUtf8SpanFormattable.TryFormat (Span<byte> utf8Destination, out int
bytesWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a half-precision floating-point number.

C#

```
public readonly struct Half : IComparable<Half>, IEquatable<Half>,
IParsable<Half>, ISpanParsable<Half>, IUtf8SpanParsable<Half>,
System.Numerics.IAdditionOperators<Half,Half,Half>,
System.Numerics.IAdditiveIdentity<Half,Half>,
System.Numerics.IBinaryFloatingPointIeee754<Half>,
System.Numerics.IBinaryNumber<Half>,
System.Numerics.IBitwiseOperators<Half,Half,Half>,
System.Numerics.IComparisonOperators<Half,Half,bool>,
System.Numerics.IDecrementOperators<Half>,
System.Numerics.IDivisionOperators<Half,Half,Half>,
System.Numerics.IEqualityOperators<Half,Half,bool>,
System.Numerics.IExponentialFunctions<Half>,
System.Numerics.IFloatingPoint<Half>,
System.Numerics.IFloatingPointConstants<Half>,
System.Numerics.IFloatingPointIeee754<Half>,
System.Numerics.IHyperbolicFunctions<Half>,
System.Numerics.IIncrementOperators<Half>,
System.Numerics.ILogarithmicFunctions<Half>,
System.Numerics.IMinMaxValue<Half>,
System.Numerics.IModulusOperators<Half,Half,Half>,
System.Numerics.IMultiplicativeIdentity<Half,Half>,
System.Numerics.IMultiplyOperators<Half,Half,Half>,
System.Numerics.INumber<Half>, System.Numerics.INumberBase<Half>,
System.Numerics.IPowerFunctions<Half>, System.Numerics.IRootFunctions<Half>,
System.Numerics.ISignedNumber<Half>,
System.Numerics.ISubtractionOperators<Half,Half,Half>,
System.Numerics.ITrigonometricFunctions<Half>,
System.Numerics.IUnaryNegationOperators<Half,Half>,
System.Numerics.IUnaryPlusOperators<Half,Half>
```

Inheritance Object → ValueType → Half

Implements IComparable , IComparable<Half> , IEquatable<Half> , IFormattable ,

ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> ,

IParsable<Half> , IParsable<TSelf> , ISpanParsable<Half> ,
ISpanParsable<TSelf> , IAdditionOperators<Half,Half,Half> ,
IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Half,Half> ,
IAdditiveIdentity<TSelf,TSelf> , IBinaryFloatingPointIeee754<Half> ,
IBinaryNumber<Half> , IBinaryNumber<TSelf> ,
IBitwiseOperators<Half,Half,Half> , IBitwiseOperators<TSelf,TSelf,TSelf> ,
IComparisonOperators<Half,Half,Boolean> ,
IComparisonOperators<TSelf,TSelf,Boolean> ,
IDecrementOperators<Half> , IDecrementOperators<TSelf> ,
IDivisionOperators<Half,Half,Half> , IDivisionOperators<TSelf,TSelf,TSelf> ,
IEqualityOperators<Half,Half,Boolean> ,
IEqualityOperators<TSelf,TOther,TResult> ,
IEqualityOperators<TSelf,TSelf,Boolean> , IExponentialFunctions<Half> ,
IExponentialFunctions<TSelf> , IFloatingPoint<Half> ,
IFloatingPoint<TSelf> , IFloatingPointConstants<Half> ,
IFloatingPointConstants<TSelf> , IFloatingPointIeee754<Half> ,
IFloatingPointIeee754<TSelf> , IHyperbolicFunctions<Half> ,
IHyperbolicFunctions<TSelf> , IIncrementOperators<Half> ,
IIncrementOperators<TSelf> , ILogarithmicFunctions<Half> ,
ILogarithmicFunctions<TSelf> , IMinMaxValue<Half> ,
IModulusOperators<Half,Half,Half> ,
IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<Half,Half> ,
IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<Half,Half,Half> ,
IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Half> , INumber<TSelf> ,
INumberBase<Half> , INumberBase<TSelf> , IPowerFunctions<Half> ,
IPowerFunctions<TSelf> , IRootFunctions<Half> , IRootFunctions<TSelf> ,
ISignedNumber<Half> , ISignedNumber<TSelf> ,
ISubtractionOperators<Half,Half,Half> ,
ISubtractionOperators<TSelf,TSelf,TSelf> , ITrigonometricFunctions<Half> ,
ITrigonometricFunctions<TSelf> , IUnaryNegationOperators<Half,Half> ,
IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<Half,Half> ,

IUnaryPlusOperators<TSelf,TSelf> , IUtf8SpanFormattable , IUtf8SpanParsable<Half> , IUtf8SpanParsable<TSelf>

Remarks

The Half value type represents a half-precision 16-bit number with values ranging from negative 65,504 to positive 65,504, as well as positive or negative zero, PositiveInfinity, NegativeInfinity, and not a number (NaN).

This is an IEEE 754-compliant `float16` type.

Properties

Expand table

E	Gets the mathematical constant `e`.
Epsilon	Represents the smallest positive Half value that is greater than zero.
MaxValue	Represents the largest possible value of Half.
MinValue	Represents the smallest possible value of Half.
MultiplicativeIdentity	Gets the multiplicative identity of the current type.
NaN	Represents not a number.
NegativeInfinity	Represents negative infinity.
NegativeOne	Gets the value `-1` for the type.
NegativeZero	Gets a value that represents negative `zero`.
One	Gets the value `1` for the type.
Pi	Gets the mathematical constant `pi`.
PositiveInfinity	Represents positive infinity.
Tau	Gets the mathematical constant `tau`.
Zero	Gets the value `0` for the type.

Methods

Expand table

Abs(Half)	Computes the absolute of a value.
Acos(Half)	Computes the arc-cosine of a value.
Acosh(Half)	Computes the hyperbolic arc-cosine of a value.
AcosPi(Half)	Computes the arc-cosine of a value and divides the result by `pi`.
Asin(Half)	Computes the arc-sine of a value.
Asinh(Half)	Computes the hyperbolic arc-sine of a value.
AsinPi(Half)	Computes the arc-sine of a value and divides the result by `pi`.
Atan(Half)	Computes the arc-tangent of a value.
Atan2(Half, Half)	Computes the arc-tangent of the quotient of two values.
Atan2Pi(Half, Half)	Computes the arc-tangent for the quotient of two values and divides the result by `pi`.
Atanh(Half)	Computes the hyperbolic arc-tangent of a value.
AtanPi(Half)	Computes the arc-tangent of a value and divides the result by pi.
BitDecrement(Half)	Decrements a value to the smallest value that compares less than a given value.
BitIncrement(Half)	Increments a value to the smallest value that compares greater than a given value.
Cbrt(Half)	Computes the cube-root of a value.
Ceiling(Half)	Computes the ceiling of a value.
Clamp(Half, Half, Half)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Half)	Compares this instance to a specified half-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified half-precision floating-point number.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is

	less than, equal to, or greater than the value of the specified object.
CopySign(Half, Half)	Copies the sign of a value to the sign of another value.
Cos(Half)	Computes the cosine of a value.
Cosh(Half)	Computes the hyperbolic cosine of a value.
CosPi(Half)	Computes the cosine of a value that has been multipled by `pi`.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DegreesToRadians(Half)	Converts a given value from degrees to radians.
Equals(Half)	Compares this instance for equality with `other`.
Equals(Object)	Returns a value that indicates whether this instance is equal to the specified `obj`.
Exp(Half)	Computes `E` raised to a given power.
Exp10(Half)	Computes `10` raised to a given power.
Exp10M1(Half)	Computes `10` raised to a given power and subtracts one.
Exp2(Half)	Computes `2` raised to a given power.
Exp2M1(Half)	Computes `2` raised to a given power and subtracts one.
ExpM1(Half)	Computes `E` raised to a given power and subtracts one.
Floor(Half)	Computes the floor of a value.
FusedMultiplyAdd(Half, Half, Half)	Computes the fused multiply-add of three values.
GetHashCode()	Returns the hash code for this instance.
Hypot(Half, Half)	Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.

Ieee754Remainder(Half, Half)	Computes the remainder of two values as specified by IEEE 754.
ILogB(Half)	Computes the integer logarithm of a value.
IsEvenInteger(Half)	Determines if a value represents an even integral number.
IsFinite(Half)	Determines whether the specified value is finite (zero, subnormal, or normal).
IsInfinity(Half)	Returns a value indicating whether the specified number evaluates to positive infinity.
IsInteger(Half)	Determines if a value represents an integral value.
IsNaN(Half)	Determines whether the specified value is not a number.
IsNegative(Half)	Determines whether the specified value is negative.
IsNegativeInfinity(Half)	Determines whether the specified value is negative infinity.
IsNormal(Half)	Determines whether the specified value is normal.
IsOddInteger(Half)	Determines if a value represents an odd integral number.
IsPositive(Half)	Determines if a value is positive.
IsPositiveInfinity(Half)	Determines whether the specified value is positive infinity.
IsPow2(Half)	Determines if a value is a power of two.
IsRealNumber(Half)	Determines if a value represents a real number.
IsSubnormal(Half)	Determines whether the specified value is subnormal.
Lerp(Half, Half, Half)	Performs a linear interpolation between two values based on the given weight.
Log(Half)	Computes the natural (`base-E` logarithm of a value.
Log(Half, Half)	Computes the logarithm of a value in the specified base.
Log10(Half)	Computes the base-10 logarithm of a value.
Log10P1(Half)	Computes the base-10 logarithm of a value plus one.
Log2(Half)	Computes the log2 of a value.
Log2P1(Half)	Computes the base-2 logarithm of a value plus one.
LogP1(Half)	Computes the natural (`base-E`) logarithm of a value plus one.

Max(Half, Half)	Compares two values to compute which is greater.
MaxMagnitude(Half, Half)	Compares two values to compute which is greater.
MaxMagnitudeNumber(Half, Half)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
MaxNumber(Half, Half)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
Min(Half, Half)	Compares two values to compute which is lesser.
MinMagnitude(Half, Half)	Compares two values to compute which is lesser.
MinMagnitudeNumber(Half, Half)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
MinNumber(Half, Half)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Parse(String)	Converts the string representation of a number to its half-precision floating-point number equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Pow(Half, Half)	Computes a value raised to a given power.
RadiansToDegrees(Half)	Converts a given value from radians to degrees.

ReciprocalEstimate(Half)	Computes an estimate of the reciprocal of a value.
ReciprocalSqrtEstimate(Half)	Computes an estimate of the reciprocal square root of a value.
RootN(Half, Int32)	Computes the n-th root of a value.
Round(Half)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Half, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Half, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Half, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
ScaleB(Half, Int32)	Computes the product of a value and its base-radix raised to the specified power.
Sign(Half)	Computes the sign of a value.
Sin(Half)	Computes the sine of a value.
SinCos(Half)	Computes the sine and cosine of a value.
SinCosPi(Half)	Computes the sine and cosine of a value that has been multiplied by `pi`.
Sinh(Half)	Computes the hyperbolic sine of a value.
SinPi(Half)	Computes the sine of a value that has been multiplied by `pi`.
Sqrt(Half)	Computes the square-root of a value.
Tan(Half)	Computes the tangent of a value.
Tanh(Half)	Computes the hyperbolic tangent of a value.
TanPi(Half)	Computes the tangent of a value that has been multipled by `pi`.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
Truncate(Half)	Truncates a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current Half instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, Half)	Tries to convert a UTF-8 character span containing the string representation of a number to its half-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Half)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Half)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Half)	Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Half)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Half)	Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Half)	Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Half)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Half)	Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Operators

Expand table

Addition(Half, Half)	Adds two values together to compute their sum.
CheckedExplicit(Half)	
Decrement(Half)	Decrements a value.
Division(Half, Half)	Divides two values together to compute their quotient.
Equality(Half, Half)	Returns a value that indicates whether two specified Half values are equal.
Explicit(Char to Half)	Explicitly converts a Char value to its nearest representable half-precision floating-point value.
Explicit(Decimal to Half)	Explicitly converts a Decimal value to its nearest representable half-precision floating-point value.
Explicit(Double to Half)	An explicit operator to convert a Double value to a Half.
Explicit(Half to Byte)	Explicitly converts a half-precision floating-point value to its nearest representable Byte value.
Explicit(Half to Char)	Explicitly converts a half-precision floating-point value to its nearest representable Char value.
Explicit(Half to Decimal)	Explicitly converts a half-precision floating-point value to its nearest representable Decimal value.
Explicit(Half to Double)	An explicit operator to convert a Half value to a Double.
Explicit(Half to Int128)	Explicitly converts a half-precision floating-point value to its nearest representable Int128.
Explicit(Half to Int16)	Explicitly converts a half-precision floating-point value to its nearest representable Int16 value.
Explicit(Half to Int32)	Explicitly converts a half-precision floating-point value to its nearest representable Int32 value.
Explicit(Half to Int64)	Explicitly converts a half-precision floating-point value to its nearest representable Int64 value.
Explicit(Half to IntPtr)	Explicitly converts a half-precision floating-point value to its nearest representable IntPtr value.
Explicit(Half to SByte)	Explicitly converts a half-precision floating-point value to its nearest representable SByte value.

Explicit(Half to Single)	An explicit operator to convert a Half value to a Single.
Explicit(Half to UInt128)	Explicitly converts a half-precision floating-point value to its nearest representable UInt128.
Explicit(Half to UInt16)	Explicitly converts a half-precision floating-point value to its nearest representable UInt16 value.
Explicit(Half to UInt32)	Explicitly converts a half-precision floating-point value to its nearest representable UInt32 value.
Explicit(Half to UInt64)	Explicitly converts a half-precision floating-point value to its nearest representable UInt64 value.
Explicit(Half to UIntPtr)	Explicitly converts a half-precision floating-point value to its nearest representable UIntPtr value.
Explicit(Int16 to Half)	Explicitly converts a Int16 value to its nearest representable half-precision floating-point value.
Explicit(Int32 to Half)	Explicitly converts a Int32 value to its nearest representable half-precision floating-point value.
Explicit(Int64 to Half)	Explicitly converts a Int64 value to its nearest representable half-precision floating-point value.
Explicit(IntPtr to Half)	Explicitly converts a IntPtr value to its nearest representable half-precision floating-point value.
Explicit(Single to Half)	An explicit operator to convert a Single value to a Half.
Explicit(UInt16 to Half)	Explicitly converts a UInt16 value to its nearest representable half-precision floating-point value.
Explicit(UInt32 to Half)	Explicitly converts a UInt32 value to its nearest representable half-precision floating-point value.
Explicit(UInt64 to Half)	Explicitly converts a UInt64 value to its nearest representable half-precision floating-point value.
Explicit(UIntPtr to Half)	Explicitly converts a UIntPtr value to its nearest representable half-precision floating-point value.
GreaterThan(Half, Half)	Returns a value that indicates whether a specified Half value is greater than another specified Half value.
GreaterThan OrEqual(Half, Half)	Returns a value that indicates whether `left` is greater than or equal to `right`.
Implicit(Byte to Half)	Implicitly converts a Byte value to its nearest representable half-precision floating-point value.

Implicit(SByte to Half)	Implicitly converts a SByte value to its nearest representable half-precision floating-point value.
Increment(Half)	Increments a value.
Inequality(Half, Half)	Returns a value that indicates whether two specified Half values are not equal.
LessThan(Half, Half)	Returns a value that indicates whether a specified Half value is less than another specified Half value.
LessThanOrEqual(Half, Half)	Returns a value that indicates whether a specified Half value is less than or equal to another specified Half value.
Modulus(Half, Half)	Divides two values together to compute their modulus or remainder.
Multiply(Half, Half)	Multiplies two values together to compute their product.
Subtraction(Half, Half)	Subtracts two values to compute their difference.
UnaryNegation(Half)	Computes the unary negation of a value.
UnaryPlus(Half)	Computes the unary plus of a value.

Explicit Interface Implementations

Expand table

IAdditiveIdentity<Half,Half>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryNumber<Half>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Half,Half,Half>.BitwiseAnd(Half, Half)	Computes the bitwise-and of two values.
IBitwiseOperators<Half,Half,Half>.BitwiseOr(Half, Half)	Computes the bitwise-or of two values.
IBitwiseOperators<Half,Half,Half>.ExclusiveOr(Half, Half)	Computes the exclusive-or of two values.
IBitwiseOperators<Half,Half,Half>.OnesComplement(Half)	Computes the ones-complement representation of a given value.
IFloatingPoint<Half>.GetExponentByteCount()	Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

IFloatingPoint<Half>.GetExponentShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current exponent.
IFloatingPoint<Half>.GetSignificandBitLength()	Gets the length, in bits, of the current significand.
IFloatingPoint<Half>.GetSignificandByteCount()	Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Half>.TryWriteExponentBigEndian(Span<Byte>, Int32)	Tries to write the current exponent, in big-endian format, to a given span.
IFloatingPoint<Half>.TryWriteExponentLittleEndian(Span<Byte>, Int32)	Tries to write the current exponent, in little-endian format, to a given span.
IFloatingPoint<Half>.TryWriteSignificandBigEndian(Span<Byte>, Int32)	Tries to write the current significand, in big-endian format, to a given span.
IFloatingPoint<Half>.TryWriteSignificandLittleEndian(Span<Byte>, Int32)	Tries to write the current significand, in little-endian format, to a given span.
INumberBase<Half>.IsCanonical(Half)	Determines if a value is in its canonical representation.
INumberBase<Half>.IsComplexNumber(Half)	Determines if a value represents a complex number.
INumberBase<Half>.IsImaginaryNumber(Half)	Determines if a value represents a pure imaginary number.
INumberBase<Half>.IsZero(Half)	Determines if a value is zero.
INumberBase<Half>.Radix	Gets the radix, or base, for the type.
INumberBase<Half>.TryConvertFromChecked<TOther>(TOther, Half)	
INumberBase<Half>.TryConvertFromSaturating<TOther>(TOther, Half)	
INumberBase<Half>.TryConvertFromTruncating<TOther>(TOther, Half)	
INumberBase<Half>.TryConvertToChecked<TOther>(Half, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Half>.TryConvertToSaturating<TOther>(Half, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

INumberBase<Half>.TryConvertToTruncating<TOther>(Half, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

See also

- Single

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.E Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `e`.

```
C#
public static Half E { get; }
```

Property Value

Half

Implements

E

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Epsilon Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest positive Half value that is greater than zero.

C#

```
public static Half Epsilon { get; }
```

Property Value

Half

5.9604645E-08

Implements

Epsilon

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of Half.

C#

```
public static Half MaxValue { get; }
```

Property Value

Half

65504

Implements

MaxValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of Half.

C#

```
public static Half MinValue { get; }
```

Property Value

Half

-65504

Implements

MinValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
public static Half MultiplicativeIdentity { get; }
```

Property Value

Half

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.NaN Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents not a number.

```
C#
public static Half NaN { get; }
```

Property Value

Half

NaN.

Implements

NaN

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.NegativeInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents negative infinity.

C#

```
public static Half NegativeInfinity { get; }
```

Property Value

Half

-Infinity.

Implements

NegativeInfinity

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
public static Half NegativeOne { get; }
```

Property Value

Half

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.NegativeZero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents negative `zero`.

C#

```
public static Half NegativeZero { get; }
```

Property Value

Half

Implements

NegativeZero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
public static Half One { get; }
```

Property Value

Half

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Pi Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `pi`.

```
C#
public static Half Pi { get; }
```

Property Value

Half

Implements

Pi

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.PositiveInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents positive infinity.

C#

```
public static Half PositiveInfinity { get; }
```

Property Value

Half

Infinity.

Implements

PositiveInfinity

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Tau Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `tau`.

```
C#
public static Half Tau { get; }
```

Property Value

Half

Implements

Tau

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
public static Half Zero { get; }
```

Property Value

Half

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Abs(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static Half Abs (Half value);
```

Parameters

`value` Half

The value for which to get its absolute.

Returns

Half

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Acos(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value.

```
C#
public static Half Acos (Half x);
```

Parameters

x Half

The value, in radians, whose arc-cosine is to be computed.

Returns

Half

The arc-cosine of `x`.

Implements

Acos(TSelf)

Remarks

This computes `arccos(x)` in the interval `[+0, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Acosh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-cosine of a value.

C#

```
public static Half Acosh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic arc-cosine is to be computed.

Returns

Half

The hyperbolic arc-cosine of `x`.

Implements

Acosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.AcosPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value and divides the result by `pi`.

```
C#
public static Half AcosPi (Half x);
```

Parameters

x Half

The value whose arc-cosine is to be computed.

Returns

Half

The arc-cosine of `x`, divided by `pi`.

Implements

AcosPi(TSelf)

Remarks

This computes `arccos(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Asin(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value.

```
C#
public static Half Asin (Half x);
```

Parameters

x Half

The value, in radians, whose arc-sine is to be computed.

Returns

Half

The arc-sine of `x`.

Implements

Asin(TSelf)

Remarks

This computes `arcsin(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Asinh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-sine of a value.

C#

```
public static Half Asinh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic arc-sine is to be computed.

Returns

Half

The hyperbolic arc-sine of `x`.

Implements

Asinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.AsinPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value and divides the result by `pi`.

```
C#
public static Half AsinPi (Half x);
```

Parameters

x Half

The value whose arc-sine is to be computed.

Returns

Half

The arc-sine of `x`, divided by `pi`.

Implements

AsinPi(TSelf)

Remarks

This computes `arcsin(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Atan(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value.

```
C#
public static Half Atan (Half x);
```

Parameters

x Half

The value, in radians, whose arc-tangent is to be computed.

Returns

Half

The arc-tangent of `x`.

Implements

Atan(TSelf)

Remarks

This computes `arctan(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Atan2(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of the quotient of two values.

C#

```
public static Half Atan2 (Half y, Half x);
```

Parameters

y Half

The y-coordinate of a point.

x Half

The x-coordinate of a point.

Returns

Half

The arc-tangent of `y` divided-by `x`.

Implements

Atan2(TSelf, TSelf)

Remarks

This computes `arctan(y / x)` in the interval `[-π, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Atan2Pi(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent for the quotient of two values and divides the result by `pi`.

```
C#
public static Half Atan2Pi (Half y, Half x);
```

Parameters

y Half

The y-coordinate of a point.

x Half

The x-coordinate of a point.

Returns

Half

The arc-tangent of `y` divided-by `x`, divided by `pi`.

Implements

Atan2Pi(TSelf, TSelf)

Remarks

This computes `arctan(y / x) / π` in the interval `[-1, +1]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Atanh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-tangent of a value.

C#

```
public static Half Atanh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic arc-tangent is to be computed.

Returns

Half

The hyperbolic arc-tangent of `x`.

Implements

Atanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.AtanPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value and divides the result by pi.

```
C#
public static Half AtanPi (Half x);
```

Parameters

x Half

The value whose arc-tangent is to be computed.

Returns

Half

The arc-tangent of `x`, divided by `pi`.

Implements

AtanPi(TSelf)

Remarks

This computes `arctan(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.BitDecrement(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value to the smallest value that compares less than a given value.

C#

```
public static Half BitDecrement (Half x);
```

Parameters

x Half

The value to be bitwise decremented.

Returns

Half

The smallest value that compares less than `x`.

Implements

BitDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.BitIncrement(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value to the smallest value that compares greater than a given value.

C#

```
public static Half BitIncrement (Half x);
```

Parameters

x Half

The value to be bitwise incremented.

Returns

Half

The smallest value that compares greater than `x`.

Implements

BitIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Cbrt(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cube-root of a value.

C#

```
public static Half Cbrt (Half x);
```

Parameters

x Half

The value whose cube-root is to be computed.

Returns

Half

The cube-root of `x`.

Implements

Cbrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Ceiling(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ceiling of a value.

C#

```
public static Half Ceiling (Half x);
```

Parameters

x Half

The value whose ceiling is to be computed.

Returns

Half

The ceiling of `x`.

Implements

Ceiling(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Clamp(Half, Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static Half Clamp (Half value, Half min, Half max);
```

Parameters

`value` Half

The value to clamp.

`min` Half

The inclusive minimum to which `value` should clamp.

`max` Half

The inclusive maximum to which `value` should clamp.

Returns

Half

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(Half)	Compares this instance to a specified half-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified half-precision floating-point number.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.

CompareTo(Half)

Compares this instance to a specified half-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified half-precision floating-point number.

C#

```
public int CompareTo (Half other);
```

Parameters

`other` Half

A half-precision floating-point number to compare.

Returns

Int32

A value less than zero if this is less than `other`, zero if this is equal to `other`, or a value greater than zero if this is greater than `other`.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.

C#

```
public int CompareTo (object? obj);
```

Parameters

`obj` Object

An object to compare, or `null`.

Returns

Int32

A value less than zero if this instance is less than `obj`, or this instance is not a number (NaN) and `obj` is a number.

-or-

Zero if this instance is equal to `obj`, or this instance and `obj` are both not a number (NaN), PositiveInfinity, or NegativeInfinity.

-or-

A value greater than zero if this instance is greater than `obj`, or this instance is a number and `obj` is not a number (NaN), or `obj` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not of type Half.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.CopySign(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
public static Half CopySign (Half value, Half sign);
```

Parameters

`value` Half

The value whose magnitude is used in the result.

`sign` Half

The value whose sign is used in the result.

Returns

Half

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Cos(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value.

```
C#
public static Half Cos (Half x);
```

Parameters

x Half

The value, in radians, whose cosine is to be computed.

Returns

Half

The cosine of `x`.

Implements

Cos(TSelf)

Remarks

This computes `cos(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Cosh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic cosine of a value.

C#

```
public static Half Cosh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic cosine is to be computed.

Returns

Half

The hyperbolic cosine of `x`.

Implements

Cosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CosPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value that has been multipled by `pi`.

```
C#
public static Half CosPi (Half x);
```

Parameters

x Half

The value, in half-revolutions, whose cosine is to be computed.

Returns

Half

The cosine of `x` multiplied-by `pi`.

Implements

CosPi(TSelf)

Remarks

This computes `cos(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static Half CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Half

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static Half CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

Half

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static Half CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

Half

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.DegreesToRadians(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from degrees to radians.

C#

```
public static Half DegreesToRadians (Half degrees);
```

Parameters

`degrees` Half

The value to convert to radians.

Returns

Half

The value of `degrees` converted to radians.

Implements

DegreesToRadians(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Half)	Compares this instance for equality with `other`.
Equals(Object)	Returns a value that indicates whether this instance is equal to the specified `obj`.

Equals(Half)

Compares this instance for equality with `other`.

C#

```
public bool Equals (Half other);
```

Parameters

`other` Half

A half-precision floating point number to compare to this instance.

Returns

Boolean

`true` if the current object is equal to `other`; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Equals(Object)

Returns a value that indicates whether this instance is equal to the specified `obj`.

```
C#
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare to this instance.

Returns

Boolean

`true` if `obj` is an instance of Half and equals the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.Exp(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power.

```
C#
public static Half Exp (Half x);
```

Parameters

x Half

The power to which `E` is raised.

Returns

Half

`E` raised to the power of `x`.

Implements

Exp(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Exp10(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power.

```
C#
public static Half Exp10 (Half x);
```

Parameters

x Half

The power to which `10` is raised.

Returns

Half

10^x

Implements

Exp10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Exp10M1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power and subtracts one.

```
C#
public static Half Exp10M1 (Half x);
```

Parameters

x Half

The power to which `10` is raised.

Returns

Half

`10ˣ - 1`

Implements

Exp10M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Exp2(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power.

```
C#
public static Half Exp2 (Half x);
```

Parameters

x Half

The power to which `2` is raised.

Returns

Half

2^x

Implements

Exp2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Exp2M1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power and subtracts one.

C#

```
public static Half Exp2M1 (Half x);
```

Parameters

x Half

The power to which `2` is raised.

Returns

Half

`2ˣ - 1`

Implements

Exp2M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ExpM1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power and subtracts one.

```
C#
public static Half ExpM1 (Half x);
```

Parameters

x Half

The power to which `E` is raised.

Returns

Half

$E^x - 1$

Implements

ExpM1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Floor(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the floor of a value.

```
C#
public static Half Floor (Half x);
```

Parameters

x Half

The value whose floor is to be computed.

Returns

Half

The floor of `x`.

Implements

Floor(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.FusedMultiplyAdd(Half, Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the fused multiply-add of three values.

```
C#
public static Half FusedMultiplyAdd (Half left, Half right, Half addend);
```

Parameters

`left` Half

The value which `right` multiplies.

`right` Half

The value which multiplies `left`.

`addend` Half

The value that is added to the product of `left` and `right`.

Returns

Half

The result of `left` times `right` plus `addend` computed as one ternary operation.

Implements

FusedMultiplyAdd(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Hypot(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.

```
C#
public static Half Hypot (Half x, Half y);
```

Parameters

x Half

The value to square and add to `y`.

y Half

The value to square and add to `x`.

Returns

Half

The square root of `x`-squared plus `y`-squared.

Implements

Hypot(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Ieee754Remainder(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the remainder of two values as specified by IEEE 754.

```
C#
public static Half Ieee754Remainder (Half left, Half right);
```

Parameters

`left` Half

The value which `right` divides.

`right` Half

The value which divides `left`.

Returns

Half

The remainder of `left` divided-by `right` as specified by IEEE 754.

Implements

Ieee754Remainder(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ILogB(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the integer logarithm of a value.

C#

```
public static int ILogB (Half x);
```

Parameters

x Half

The value whose integer logarithm is to be computed.

Returns

Int32

The integer logarithm of `x`.

Implements

ILogB(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsEvenInteger(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsFinite(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is finite (zero, subnormal, or normal).

C#

```
public static bool IsFinite (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if the specified value is finite (zero, subnormal or normal); otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsInfinity(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to positive infinity.

C#

```
public static bool IsInfinity (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` evaluates to PositiveInfinity; otherwise `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsInteger(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral value.

```
C#
public static bool IsInteger (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` and `3.0` will return `true` while `2.2` and `3.3` will return `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsNaN(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is not a number.

C#

```
public static bool IsNaN (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` evaluates to not a number NaN; otherwise `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsNegative(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is negative.

C#

```
public static bool IsNegative (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if negative; `false` otherwise.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsNegativeInfinity(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is negative infinity.

C#

```
public static bool IsNegativeInfinity (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` evaluates to NegativeInfinity; otherwise `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsNormal(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is normal.

C#

```
public static bool IsNormal (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` is normal; `false` otherwise.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsOddInteger(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (Half value);
```

Parameters

value Half

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsPositive(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsPositiveInfinity(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is positive infinity.

C#

```
public static bool IsPositiveInfinity (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` evaluates to PositiveInfinity; otherwise `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsPow2(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsRealNumber(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
public static bool IsRealNumber (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Remarks

This function returns `true` for a complex number `a + bi` where `b` is zero.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IsSubnormal(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is subnormal.

C#

```
public static bool IsSubnormal (Half value);
```

Parameters

`value` Half

A Half floating-point number.

Returns

Boolean

`true` if `value` is subnormal; `false` `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Lerp(Half, Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs a linear interpolation between two values based on the given weight.

C#

```
public static Half Lerp (Half value1, Half value2, Half amount);
```

Parameters

`value1` Half

The first value, which is intended to be the lower bound.

`value2` Half

The second value, which is intended to be the upper bound.

`amount` Half

A value, intended to be between 0 and 1, that indicates the weight of the interpolation.

Returns

Half

The interpolated value.

Implements

Lerp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Log Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Log(Half)	Computes the natural (`base-E` logarithm of a value.
Log(Half, Half)	Computes the logarithm of a value in the specified base.

Log(Half)

Computes the natural (`base-E` logarithm of a value.

```
C#
public static Half Log (Half x);
```

Parameters

x Half

The value whose natural logarithm is to be computed.

Returns

Half

The natural logarithm of `x`.

Implements

Log(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Log(Half, Half)

Computes the logarithm of a value in the specified base.

C#

```
public static Half Log (Half x, Half newBase);
```

Parameters

`x` Half

The value whose logarithm is to be computed.

`newBase` Half

The base in which the logarithm is to be computed.

Returns

Half

The base-`newBase` logarithm of `x`.

Implements

Log(TSelf, TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Log10(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value.

C#

```
public static Half Log10 (Half x);
```

Parameters

x Half

The value whose base-10 logarithm is to be computed.

Returns

Half

The base-10 logarithm of `x`.

Implements

Log10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Log10P1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value plus one.

C#

```
public static Half Log10P1 (Half x);
```

Parameters

x Half

The value to which one is added before computing the base-10 logarithm.

Returns

Half

$\log_{10}(x + 1)$

Implements

Log10P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Log2(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static Half Log2 (Half value);
```

Parameters

`value` Half

The value whose log2 is to be computed.

Returns

Half

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Log2P1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-2 logarithm of a value plus one.

C#

```
public static Half Log2P1 (Half x);
```

Parameters

x Half

The value to which one is added before computing the base-2 logarithm.

Returns

Half

$\log_2(x + 1)$

Implements

Log2P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.LogP1(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the natural (`base-E`) logarithm of a value plus one.

```
C#
public static Half LogP1 (Half x);
```

Parameters

x Half

The value to which one is added before computing the natural logarithm.

Returns

Half

$\log_e(x + 1)$

Implements

LogP1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Max(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static Half Max (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MaxMagnitude(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static Half MaxMagnitude (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MaxMagnitudeNumber(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#
public static Half MaxMagnitudeNumber (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.MaxNumber(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
public static Half MaxNumber (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Min(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static Half Min (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MinMagnitude(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static Half MinMagnitude (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.MinMagnitudeNumber(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#
public static Half MinMagnitudeNumber (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.MinNumber(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
public static Half MinNumber (Half x, Half y);
```

Parameters

x Half

The value to compare with `y`.

y Half

The value to compare with `x`.

Returns

Half

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String)	Converts the string representation of a number to its half-precision floating-point number equivalent.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.

Parse(String)

Converts the string representation of a number to its half-precision floating-point number equivalent.

```
C#

public static Half Parse (string s);
```

Parameters

s String

A string that contains a number to convert.

Returns

Half

A half-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```C#
public static Half Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Half

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```C#
public static Half Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Half

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.

```
C#

public static Half Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

Returns

Half

A half-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

ArgumentException

`style` is not a NumberStyles value.

-or-

NumberStyles includes the AllowHexSpecifier value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.

```
C#
```

```
public static Half Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Half

A half-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static Half Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Half

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.

C#

```
public static Half Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Half

A half-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Exceptions

FormatException

`s` does not represent a number in a valid format.

ArgumentException
`style` is not a NumberStyles value.

-or-

NumberStyles includes the AllowHexSpecifier value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.

C#

```
public static Half Parse (string s, System.Globalization.NumberStyles
style = System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`s` String
A string that contains a number to convert.

`style` NumberStyles
A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider
An object that supplies culture-specific formatting information about `s`.

Returns

Half

A half-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

ArgumentException

`style` is not a NumberStyles value.

-or-

NumberStyles includes the AllowHexSpecifier value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.Pow(Half, Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes a value raised to a given power.

```
C#
public static Half Pow (Half x, Half y);
```

Parameters

x Half

The value which is raised to the power of `x`.

y Half

The power to which `x` is raised.

Returns

Half

`x` raised to the power of `y`.

Implements

Pow(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.RadiansToDegrees(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from radians to degrees.

C#

```
public static Half RadiansToDegrees (Half radians);
```

Parameters

`radians` Half

The value to convert to degrees.

Returns

Half

The value of `radians` converted to degrees.

Implements

RadiansToDegrees(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ReciprocalEstimate(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal of a value.

C#

```
public static Half ReciprocalEstimate (Half x);
```

Parameters

x Half

The value whose estimate of the reciprocal is to be computed.

Returns

Half

An estimate of the reciprocal of `x`.

Implements

ReciprocalEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ReciprocalSqrtEstimate(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal square root of a value.

```
C#
public static Half ReciprocalSqrtEstimate (Half x);
```

Parameters

x Half

The value whose estimate of the reciprocal square root is to be computed.

Returns

Half

An estimate of the reciprocal square root of `x`.

Implements

ReciprocalSqrtEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.RootN(Half, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the n-th root of a value.

```
C#
public static Half RootN (Half x, int n);
```

Parameters

x Half

The value whose `n`-th root is to be computed.

n Int32

The degree of the root to be computed.

Returns

Half

The `n`-th root of `x`.

Implements

RootN(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Round(Half)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Half, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Half, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
Round(Half, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

Round(Half)

Rounds a value to the nearest integer using the default rounding mode (ToEven).

C#

```
public static Half Round (Half x);
```

Parameters

x Half

The value to round.

Returns

Half

The result of rounding `x` to the nearest integer using the default rounding mode.

Implements

Round(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Half, Int32)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

C#

```
public static Half Round (Half x, int digits);
```

Parameters

`x` Half

The value to round.

`digits` Int32

The number of fractional digits to which `x` should be rounded.

Returns

Half

The result of rounding `x` to `digits` fractional-digits using the default rounding mode.

Implements

Round(TSelf, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Half, MidpointRounding)

Rounds a value to the nearest integer using the specified rounding mode.

C#

```
public static Half Round (Half x, MidpointRounding mode);
```

Parameters

`x` Half

The value to round.

`mode` MidpointRounding

The mode under which `x` should be rounded.

Returns

Half

The result of rounding `x` to the nearest integer using `mode`.

Implements

Round(TSelf, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Half, Int32, MidpointRounding)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

```
C#

public static Half Round (Half x, int digits, MidpointRounding mode);
```

Parameters

x Half

The value to round.

digits Int32

The number of fractional digits to which `x` should be rounded.

mode MidpointRounding

The mode under which `x` should be rounded.

Returns

Half

The result of rounding `x` to `digits` fractional-digits using `mode`.

Implements

Round(TSelf, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ScaleB(Half, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the product of a value and its base-radix raised to the specified power.

```
C#
public static Half ScaleB (Half x, int n);
```

Parameters

x Half

The value which base-radix raised to the power of `n` multiplies.

n Int32

The value to which base-radix is raised before multipliying `x`.

Returns

Half

The product of `x` and base-radix raised to the power of `n`.

Implements

ScaleB(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Sign(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (Half value);
```

Parameters

`value` Half

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Sin(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value.

```
C#
public static Half Sin (Half x);
```

Parameters

x Half

The value, in radians, whose sine is to be computed.

Returns

Half

The sine of `x`.

Implements

Sin(TSelf)

Remarks

This computes `sin(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.SinCos(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value.

```
C#
public static (Half Sin, Half Cos) SinCos (Half x);
```

Parameters

x Half

The value, in radians, whose sine and cosine are to be computed.

Returns

ValueTuple<Half,Half>

The sine and cosine of `x`.

Implements

SinCos(TSelf)

Remarks

This computes `(sin(x), cos(x))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.SinCosPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value that has been multiplied by `pi`.

```
C#
public static (Half SinPi, Half CosPi) SinCosPi (Half x);
```

Parameters

x Half

The value, in half-revolutions, that is multipled by `pi` before computing its sine and cosine.

Returns

ValueTuple<Half,Half>

The sine and cosine of `x` multiplied-by `pi`.

Implements

SinCosPi(TSelf)

Remarks

This computes `(sin(x * π), cos(x * π))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Sinh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic sine of a value.

C#

```
public static Half Sinh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic sine is to be computed.

Returns

Half

The hyperbolic sine of `x`.

Implements

Sinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.SinPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value that has been multiplied by `pi`.

```
C#
public static Half SinPi (Half x);
```

Parameters

x Half

The value, in half-revolutions, that is multipled by `pi` before computing its sine.

Returns

Half

The sine of `x` multiplied-by `pi`.

Implements

SinPi(TSelf)

Remarks

This computes `sin(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Sqrt(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the square-root of a value.

C#

```
public static Half Sqrt (Half x);
```

Parameters

x Half

The value whose square-root is to be computed.

Returns

Half

The square-root of `x`.

Implements

Sqrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Tan(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value.

```
C#
public static Half Tan (Half x);
```

Parameters

x Half

The value, in radians, whose tangent is to be computed.

Returns

Half

The tangent of `x`.

Implements

Tan(TSelf)

Remarks

This computes `tan(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Tanh(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic tangent of a value.

C#

```
public static Half Tanh (Half x);
```

Parameters

x Half

The value, in radians, whose hyperbolic tangent is to be computed.

Returns

Half

The hyperbolic tangent of `x`.

Implements

Tanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.TanPi(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value that has been multipled by `pi`.

```
C#
public static Half TanPi (Half x);
```

Parameters

x Half

The value, in half-revolutions, that is multipled by `pi` before computing its tangent.

Returns

Half

The tangent of `x` multiplied-by `pi`.

Implements

TanPi(TSelf)

Remarks

This computes `tan(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid.

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(String, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Truncate(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Truncates a value.

C#

```
public static Half Truncate (Half x);
```

Parameters

x Half

The value to truncate.

Returns

Half

The truncation of `x`.

Implements

Truncate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current Half instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current Half instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful, otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
```

```
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, Half)	Tries to convert a UTF-8 character span containing the string representation of a number to its half-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Char>, Half)	Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Half)	Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Half)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Half)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Half)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Half)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Half)	Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Half)	Converts the string representation of a number to its half-precision floating-point number equivalent. A

	return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, Half)

Tries to convert a UTF-8 character span containing the string representation of a number to its half-precision floating-point number equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out Half
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A read-only UTF-8 character span that contains the number to convert.

`result` Half

When this method returns, contains a half-precision floating-point number equivalent of the numeric value or symbol contained in `utf8Text` if the conversion succeeded or zero if the conversion failed. The conversion fails if the `utf8Text` is Empty or is not in a valid format. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Half)

Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out Half result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span that contains a number to convert.

`result` Half

When this method returns, contains the half-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or a default Half value if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty or is not a number in a valid format. If `s` is a valid number less than MinValue, `result` contains NegativeInfinity. If `s` is a valid number greater than MaxValue, `result` contains PositiveInfinity. This parameter is treated as uninitialized.

Returns

Boolean

`true` if `s` was converted successfully, `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(String, Half)

Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (string? s, out Half result);
```

Parameters

`s` String

A string that contains a number to convert.

`result` Half

When this method returns, contains the half-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or a default Half value if the conversion failed. The conversion fails if `s` is `null` or Empty or is not a number in a valid format. If `s` is a valid number less than MinValue, `result` contains NegativeInfinity. If `s` is a valid number greater than MaxValue, `result` contains PositiveInfinity. This parameter is treated as uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Half)

Tries to parse a span of UTF-8 characters into a value.

```
C#
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out Half result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Half

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, Half)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out Half result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Half

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Half)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
Half result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Half

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Half)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
Half result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Half

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Half)

Converts the span representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
Half result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Half

When this method returns, contains the half-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or a default Half value if the conversion failed. The conversion fails if `s` is not a number in a valid format. If `s` is a valid number less than MinValue, `result` contains NegativeInfinity. If `s` is a valid number greater than MaxValue, `result` contains PositiveInfinity. This parameter is treated as uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

NumberStyles is the AllowHexSpecifier value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(String, NumberStyles, IFormatProvider, Half)

Converts the string representation of a number to its half-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
```

```
  style, IFormatProvider? provider, out Half result);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Half

When this method returns, contains the half-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or a default Half value if the conversion failed. The conversion fails if `s` is `null` or Empty or is not a number in a valid format. If `s` is a valid number less than MinValue, `result` contains NegativeInfinity. If `s` is a valid number greater than MaxValue, `result` contains PositiveInfinity. This parameter is treated as uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

NumberStyles is the AllowHexSpecifier value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Addition(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

C#

```
public static Half operator + (Half left, Half right);
```

Parameters

`left` Half

The value to which `right` is added.

`right` Half

The value which is added to `left`.

Returns

Half

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.CheckedExplicit(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
public static byte op_CheckedExplicit (Half value);
```

Parameters

`value` Half

Returns

Byte

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.Decrement(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

C#

```
public static Half operator -- (Half value);
```

Parameters

`value` Half

The value to decrement.

Returns

Half

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Division(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their quotient.

C#

```
public static Half operator / (Half left, Half right);
```

Parameters

`left` Half

The value which `right` divides.

`right` Half

The value which divides `left`.

Returns

Half

The quotient of `left` divided-by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Equality(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Half values are equal.

```
C#
public static bool operator == (Half left, Half right);
```

Parameters

`left` Half

The first value to compare.

`right` Half

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are equal; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Explicit(Half to UInt32)	Explicitly converts a half-precision floating-point value to its nearest representable UInt32 value.
Explicit(Half to UInt64)	Explicitly converts a half-precision floating-point value to its nearest representable UInt64 value.
Explicit(Half to UIntPtr)	Explicitly converts a half-precision floating-point value to its nearest representable UIntPtr value.
Explicit(Int16 to Half)	Explicitly converts a Int16 value to its nearest representable half-precision floating-point value.
Explicit(Int32 to Half)	Explicitly converts a Int32 value to its nearest representable half-precision floating-point value.
Explicit(Int64 to Half)	Explicitly converts a Int64 value to its nearest representable half-precision floating-point value.
Explicit(UInt64 to Half)	Explicitly converts a UInt64 value to its nearest representable half-precision floating-point value.
Explicit(Single to Half)	An explicit operator to convert a Single value to a Half.
Explicit(UInt16 to Half)	Explicitly converts a UInt16 value to its nearest representable half-precision floating-point value.
Explicit(UInt32 to Half)	Explicitly converts a UInt32 value to its nearest representable half-precision floating-point value.
Explicit(Half to UInt16)	Explicitly converts a half-precision floating-point value to its nearest representable UInt16 value.

Explicit(UIntPtr to Half)	Explicitly converts a UIntPtr value to its nearest representable half-precision floating-point value.
Explicit(IntPtr to Half)	Explicitly converts a IntPtr value to its nearest representable half-precision floating-point value.
Explicit(Half to UInt128)	Explicitly converts a half-precision floating-point value to its nearest representable UInt128.
Explicit(Half to Single)	An explicit operator to convert a Half value to a Single.
Explicit(Half to SByte)	Explicitly converts a half-precision floating-point value to its nearest representable SByte value.
Explicit(Char to Half)	Explicitly converts a Char value to its nearest representable half-precision floating-point value.
Explicit(Double to Half)	An explicit operator to convert a Double value to a Half.
Explicit(Half to Byte)	Explicitly converts a half-precision floating-point value to its nearest representable Byte value.
Explicit(Half to Char)	Explicitly converts a half-precision floating-point value to its nearest representable Char value.
Explicit(Half to Decimal)	Explicitly converts a half-precision floating-point value to its nearest representable Decimal value.
Explicit(Decimal to Half)	Explicitly converts a Decimal value to its nearest representable half-precision floating-point value.
Explicit(Half to Int128)	Explicitly converts a half-precision floating-point value to its nearest representable Int128.
Explicit(Half to Int16)	Explicitly converts a half-precision floating-point value to its nearest representable Int16 value.
Explicit(Half to Int32)	Explicitly converts a half-precision floating-point value to its nearest representable Int32 value.
Explicit(Half to Int64)	Explicitly converts a half-precision floating-point value to its nearest representable Int64 value.
Explicit(Half to IntPtr)	Explicitly converts a half-precision floating-point value to its nearest representable IntPtr value.
Explicit(Half to Double)	An explicit operator to convert a Half value to a Double.

Explicit(Half to UInt32)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable UInt32 value.

```C#
[System.CLSCompliant(false)]
public static explicit operator uint (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

UInt32

`value` converted to its nearest representable UInt32 value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to UInt64)

> Important

This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable UInt64 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator ulong (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

UInt64

`value` converted to its nearest representable UInt64 value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to UIntPtr)

Important

This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable UIntPtr value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator UIntPtr (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

UIntPtr

`value` converted to its nearest representable UIntPtr value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int16 to Half)

Explicitly converts a Int16 value to its nearest representable half-precision floating-point value.

```
C#

public static explicit operator Half (short value);
```

Parameters

`value` Int16

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int32 to Half)

Explicitly converts a Int32 value to its nearest representable half-precision floating-point value.

C#

```
public static explicit operator Half (int value);
```

Parameters

`value` Int32

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int64 to Half)

Explicitly converts a Int64 value to its nearest representable half-precision floating-point value.

```
C#

public static explicit operator Half (long value);
```

Parameters

`value` Int64

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt64 to Half)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a UInt64 value to its nearest representable half-precision floating-point value.

```
C#

[System.CLSCompliant(false)]
```

```
public static explicit operator Half (ulong value);
```

Parameters

`value` UInt64

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Single to Half)

An explicit operator to convert a Single value to a Half.

C#

```
public static explicit operator Half (float value);
```

Parameters

`value` Single

The single-precision floating point value to convert to Half.

Returns

Half

The Half representation of the specified single-precision floating point `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Explicit(UInt16 to Half)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a UInt16 value to its nearest representable half-precision floating-point value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator Half (ushort value);
```

Parameters

`value` UInt16

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt32 to Half)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a UInt32 value to its nearest representable half-precision floating-point value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator Half (uint value);
```

Parameters

`value` UInt32

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to UInt16)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable UInt16 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator ushort (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

UInt16

`value` converted to its nearest representable UInt16 value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UIntPtr to Half)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a UIntPtr value to its nearest representable half-precision floating-point value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator Half (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(IntPtr to Half)

Explicitly converts a IntPtr value to its nearest representable half-precision floating-point value.

```
C#

public static explicit operator Half (IntPtr value);
```

Parameters

`value` IntPtr

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to UInt128)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable UInt128.

C#

```
[System.CLSCompliant(false)]
public static explicit operator UInt128 (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Single)

An explicit operator to convert a Half value to a Single.

```
C#
public static explicit operator float (Half value);
```

Parameters

`value` Half

The half-precision floating point value to convert to Single.

Returns

Single

The Single representation of the specified half-precision floating point `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Explicit(Half to SByte)

Important

This API is not CLS-compliant.

Explicitly converts a half-precision floating-point value to its nearest representable SByte value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator sbyte (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

SByte

`value` converted to its nearest representable SByte value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Char to Half)

Explicitly converts a Char value to its nearest representable half-precision floating-point value.

```
C#

public static explicit operator Half (char value);
```

Parameters

`value` Char

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Double to Half)

An explicit operator to convert a Double value to a Half.

```
C#
public static explicit operator Half (double value);
```

Parameters

`value` Double

The double-precision floating point value to convert to Half.

Returns

Half

The Half representation of the specified double-precision floating point `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Explicit(Half to Byte)

Explicitly converts a half-precision floating-point value to its nearest representable Byte value.

```
C#
public static explicit operator byte (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Byte

`value` converted to its nearest representable Byte value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Char)

Explicitly converts a half-precision floating-point value to its nearest representable Char value.

```
C#
public static explicit operator char (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Char

`value` converted to its nearest representable Char value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Decimal)

Explicitly converts a half-precision floating-point value to its nearest representable Decimal value.

```
C#
public static explicit operator decimal (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Decimal

`value` converted to its nearest representable Decimal value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Decimal to Half)

Explicitly converts a Decimal value to its nearest representable half-precision floating-point value.

```
C#
public static explicit operator Half (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Int128)

Explicitly converts a half-precision floating-point value to its nearest representable Int128.

```
C#
public static explicit operator Int128 (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Int16)

Explicitly converts a half-precision floating-point value to its nearest representable Int16 value.

```
C#
public static explicit operator short (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Int16

`value` converted to its nearest representable Int16 value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Int32)

Explicitly converts a half-precision floating-point value to its nearest representable Int32 value.

```
C#
public static explicit operator int (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Int32

`value` converted to its nearest representable Int32 value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Int64)

Explicitly converts a half-precision floating-point value to its nearest representable Int64 value.

```
C#
public static explicit operator long (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

Int64

`value` converted to its nearest representable Int64 value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to IntPtr)

Explicitly converts a half-precision floating-point value to its nearest representable IntPtr value.

```
C#
public static explicit operator IntPtr (Half value);
```

Parameters

`value` Half

The value to convert.

Returns

IntPtr

`value` converted to its nearest representable IntPtr value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Half to Double)

An explicit operator to convert a Half value to a Double.

```
C#
public static explicit operator double (Half value);
```

Parameters

`value` Half

The half-precision floating point value to convert to Double.

Returns

Double

The Double representation of the specified half-precision floating point `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.GreaterThan(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Half value is greater than another specified Half value.

```
C#
public static bool operator > (Half left, Half right);
```

Parameters

`left` Half

The first value to compare.

`right` Half

The second value to compare.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.GreaterThanOrEqual(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether `left` is greater than or equal to `right`.

```
C#
public static bool operator >= (Half left, Half right);
```

Parameters

`left` Half

The first value to compare.

`right` Half

The second value to compare.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(SByte to Half)	Implicitly converts a SByte value to its nearest representable half-precision floating-point value.
Implicit(Byte to Half)	Implicitly converts a Byte value to its nearest representable half-precision floating-point value.

Implicit(SByte to Half)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a SByte value to its nearest representable half-precision floating-point value.

C#

```
[System.CLSCompliant(false)]
public static implicit operator Half (sbyte value);
```

Parameters

`value` SByte

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Byte to Half)

Implicitly converts a Byte value to its nearest representable half-precision floating-point value.

```
C#
public static implicit operator Half (byte value);
```

Parameters

`value` Byte

The value to convert.

Returns

Half

`value` converted to its nearest representable half-precision floating-point value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Increment(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

C#

```
public static Half operator ++ (Half value);
```

Parameters

`value` Half

The value to increment.

Returns

Half

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Inequality(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Half values are not equal.

```
C#
public static bool operator != (Half left, Half right);
```

Parameters

`left` Half

The first value to compare.

`right` Half

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are not equal; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.LessThan(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Half value is less than another specified Half value.

```
C#
public static bool operator < (Half left, Half right);
```

Parameters

left Half

The first value to compare.

right Half

The second value to compare.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.LessThanOrEqual(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Half value is less than or equal to another specified Half value.

C#

```
public static bool operator <= (Half left, Half right);
```

Parameters

`left` Half

The first value to compare.

`right` Half

The second value to compare.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Modulus(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#
public static Half operator % (Half left, Half right);
```

Parameters

`left` Half

The value which `right` divides.

`right` Half

The value which divides `left`.

Returns

Half

The modulus or remainder of `left` divided-by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Multiply(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#
public static Half operator * (Half left, Half right);
```

Parameters

`left` Half

The value which `right` multiplies.

`right` Half

The value which multiplies `left`.

Returns

Half

The product of `left` multiplied-by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.Subtraction(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#
public static Half operator - (Half left, Half right);
```

Parameters

`left` Half

The value from which `right` is subtracted.

`right` Half

The value which is subtracted from `left`.

Returns

Half

The difference of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.UnaryNegation(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
public static Half operator - (Half value);
```

Parameters

`value` Half

The value for which to compute the unary negation.

Returns

Half

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.UnaryPlus(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
public static Half operator + (Half value);
```

Parameters

`value` Half

The value for which to compute the unary plus.

Returns

Half

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IAdditiveIdentity<Half,Half>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

```
C#

static Half
System.Numerics.IAdditiveIdentity<System.Half,System.Half>.AdditiveIdentity
{ get; }
```

Property Value

Half

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.IBinaryNumber<Half>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static Half System.Numerics.IBinaryNumber<System.Half>.AllBitsSet { get; }
```

Property Value

Half

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IBitwiseOperators<Half,Half,Half>.BitwiseAnd(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static Half IBitwiseOperators<Half,Half,Half>.operator & (Half left, Half
right);
```

Parameters

`left` Half

The value to `and` with `right`.

`right` Half

The value to `and` with `left`.

Returns

Half

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IBitwise Operators<Half,Half,Half>.Bitwise Or(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static Half IBitwiseOperators<Half,Half,Half>.operator | (Half left, Half
right);
```

Parameters

left Half

The value to `or` with `right`.

right Half

The value to `or` with `left`.

Returns

Half

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IBitwiseOperators<Half,Half,Half>.ExclusiveOr(Half, Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static Half IBitwiseOperators<Half,Half,Half>.operator ^ (Half left, Half
right);
```

Parameters

`left` Half

The value to xor with `right`.

`right` Half

The value to xor with `left`.

Returns

Half

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IBitwiseOperators<Half,Half,Half>.OnesComplement(Half) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static Half IBitwiseOperators<Half,Half,Half>.operator ~ (Half value);
```

Parameters

`value` Half

The value for which to compute the ones-complement.

Returns

Half

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IFloatingPoint<Half>.GetExponentByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

```
C#

int IFloatingPoint<Half>.GetExponentByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

Implements

GetExponentByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.IFloatingPoint<Half>.GetExponentShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current exponent.

C#

```
int IFloatingPoint<Half>.GetExponentShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current exponent.

Implements

GetExponentShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.IFloatingPoint<Half>.GetSignificandBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the current significand.

C#

```
int IFloatingPoint<Half>.GetSignificandBitLength ();
```

Returns

Int32

The length, in bits, of the current significand.

Implements

GetSignificandBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Half.IFloatingPoint<Half>.GetSignificandByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<Half>.GetSignificandByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

Implements

GetSignificandByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.IFloatingPoint<Half>.TryWriteExponentBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<Half>.TryWriteExponentBigEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IFloatingPoint<Half>.TryWriteExponentLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<Half>.TryWriteExponentLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IFloatingPoint<Half>.TryWriteSignificandBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<Half>.TryWriteSignificandBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.IFloatingPoint<Half>.TryWriteSignificandLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<Half>.TryWriteSignificandLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.IsCanonical(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<Half>.IsCanonical (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.IsComplexNumber(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<Half>.IsComplexNumber (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.IsImaginaryNumber(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<Half>.IsImaginaryNumber (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.IsZero(Half) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<Half>.IsZero (Half value);
```

Parameters

`value` Half

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Half>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Half>.TryConvertFromChecked<TOther> (TOther value,
out Half result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Half

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.INumberBase<Half>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Half>.TryConvertFromSaturating<TOther> (TOther
value, out Half result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Half

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.INumberBase<Half>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Half>.TryConvertFromTruncating<TOther> (TOther
value, out Half result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Half

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Half.INumberBase<Half>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<Half>.TryConvertToChecked<TOther> (Half value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Half

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Half.INumberBase<Half>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<Half>.TryConvertToSaturating<TOther> (Half value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Half

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Half.INumberBase<Half>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<Half>.TryConvertToTruncating<TOther> (Half value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Half

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

HashCode Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Combines the hash code for multiple values into a single hash code.

C#

```
public struct HashCode
```

Inheritance Object → ValueType → HashCode

Examples

The static methods in this class combine the default hash codes of *up to eight* values.

C#

```
using System;
using System.Collections.Generic;

public struct OrderOrderLine : IEquatable<OrderOrderLine>
{
    public int OrderId { get; }
    public int OrderLineId { get; }

    public OrderOrderLine(int orderId, int orderLineId) => (OrderId,
OrderLineId) = (orderId, orderLineId);

    public override bool Equals(object obj) => obj is OrderOrderLine o &&
Equals(o);

    public bool Equals(OrderOrderLine other) => OrderId == other.OrderId &&
OrderLineId == other.OrderLineId;

    public override int GetHashCode() => HashCode.Combine(OrderId,
OrderLineId);
}

class Program
{
```

```
    static void Main(string[] args)
    {
        var set = new HashSet<OrderOrderLine>
        {
            new OrderOrderLine(1, 1),
            new OrderOrderLine(1, 1),
            new OrderOrderLine(1, 2)
        };

        Console.WriteLine($"Item count: {set.Count}.");
    }
}
// The example displays the following output:
// Item count: 2.
```

> ⓘ **Important**
>
> **ToHashCode()** must be called at most once per instance of **HashCode**.

The instance methods in this class combine the hash codes of *more than eight* values.

C#

```
using System;
using System.Collections.Generic;

public struct Path : IEquatable<Path>
{
    public IReadOnlyList<string> Segments { get; }

    public Path(params string[] segments) => Segments = segments;

    public override bool Equals(object obj) => obj is Path o && Equals(o);

    public bool Equals(Path other)
    {
        if (ReferenceEquals(Segments, other.Segments)) return true;
        if (Segments is null || other.Segments is null) return false;
        if (Segments.Count != other.Segments.Count) return false;

        for (var i = 0; i < Segments.Count; i++)
        {
            if (!string.Equals(Segments[i], other.Segments[i]))
                return false;
        }

        return true;
    }

    public override int GetHashCode()
    {
```

```
        var hash = new HashCode();

        for (var i = 0; i < Segments?.Count; i++)
            hash.Add(Segments[i]);

        return hash.ToHashCode();
    }
}

class Program
{
    static void Main(string[] args)
    {
        var set = new HashSet<Path>
        {
            new Path("C:", "tmp", "file.txt"),
            new Path("C:", "tmp", "file.txt"),
            new Path("C:", "tmp", "file.tmp")
        };

        Console.WriteLine($"Item count: {set.Count}.");
    }
}
// The example displays the following output:
// Item count: 2.
```

The instance methods also combine the hash codes produced by a specific IEqualityComparer<T> implementation.

C#

```
using System;
using System.Collections.Generic;

public struct Path : IEquatable<Path>
{
    public IReadOnlyList<string> Segments { get; }

    public Path(params string[] segments) => Segments = segments;

    public override bool Equals(object obj) => obj is Path o && Equals(o);

    public bool Equals(Path other)
    {
        if (ReferenceEquals(Segments, other.Segments)) return true;
        if (Segments is null || other.Segments is null) return false;
        if (Segments.Count != other.Segments.Count) return false;

        for (var i = 0; i < Segments.Count; i++)
        {
            if (!string.Equals(Segments[i], other.Segments[i],
StringComparison.OrdinalIgnoreCase))
                return false;
```

```
        }

        return true;
    }

    public override int GetHashCode()
    {
        var hash = new HashCode();

        for (var i = 0; i < Segments?.Count; i++)
            hash.Add(Segments[i], StringComparer.OrdinalIgnoreCase);

        return hash.ToHashCode();
    }
}

class Program
{
    static void Main(string[] args)
    {
        var set = new HashSet<Path>
        {
            new Path("C:", "tmp", "file.txt"),
            new Path("C:", "TMP", "file.txt"),
            new Path("C:", "tmp", "FILE.TXT")
        };

        Console.WriteLine($"Item count: {set.Count}.");
    }
}
// The example displays the following output:
// Item count: 1.
```

The HashCode structure must be passed by-reference to other methods, as it is a value type.

C#

```
using System;
using System.Collections.Generic;

public struct Path : IEquatable<Path>
{
    public IReadOnlyList<string> Segments { get; }

    public Path(params string[] segments) => Segments = segments;

    public override bool Equals(object obj) => obj is Path o && Equals(o);

    public bool Equals(Path other)
    {
        if (ReferenceEquals(Segments, other.Segments)) return true;
        if (Segments is null || other.Segments is null) return false;
```

```
        if (Segments.Count != other.Segments.Count) return false;

        for (var i = 0; i < Segments.Count; i++)
        {
            if (!PlatformUtils.PathEquals(Segments[i], other.Segments[i]))
                return false;
        }

        return true;
    }

    public override int GetHashCode()
    {
        var hash = new HashCode();

        for (var i = 0; i < Segments?.Count; i++)
            PlatformUtils.AddPath(ref hash, Segments[i]);

        return hash.ToHashCode();
    }
}

internal static class PlatformUtils
{
    public static bool PathEquals(string a, string b) => string.Equals(a, b,
StringComparison.OrdinalIgnoreCase);
    public static void AddPath(ref HashCode hash, string path) =>
hash.Add(path, StringComparer.OrdinalIgnoreCase);
}

class Program
{
    static void Main(string[] args)
    {
        var set = new HashSet<Path>
        {
            new Path("C:", "tmp", "file.txt"),
            new Path("C:", "TMP", "file.txt"),
            new Path("C:", "tmp", "FILE.TXT")
        };

        Console.WriteLine($"Item count: {set.Count}.");
    }
}
// The example displays the following output:
// Item count: 1.
```

Remarks

You can use HashCode to combine multiple values (for example, fields on a structure or class) into a single hash code. This structure has static and instance methods that

operate differently:

- The static methods accept a set of up to eight values to combine.
- Two instance methods operate in a streaming fashion, accepting values one at a time.

> ⚠ **Warning**
>
> It's best-practice to consider hash codes as an implementation detail, as the implementation may change across assembly versions. Do not store hash codes produced by **HashCode** in serialized structures, for example, on-disk. **HashCode** uses a statically initialized random seed to enforce this best practice, meaning that the hash codes are only deterministic within the scope of an operating system process.

Methods

Expand table

Add<T>(T)	Adds a single value to the hash code.
Add<T>(T, IEqualityComparer<T>)	Adds a single value to the hash code, specifying the type that provides the hash code function.
AddBytes(ReadOnlySpan<Byte>)	Adds a span of bytes to the hash code.
Combine<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Combines eight values into a hash code.
Combine<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Combines seven values into a hash code.
Combine<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Combines six values into a hash code.
Combine<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Combines five values into a hash code.
Combine<T1,T2,T3,T4>(T1, T2, T3, T4)	Combines four values into a hash code.
Combine<T1,T2,T3>(T1, T2, T3)	Combines three values into a hash code.
Combine<T1,T2>(T1, T2)	Combines two values into a hash code.
Combine<T1>(T1)	Diffuses the hash code returned by the specified value.

Equals(Object)	**Obsolete.** This method is not supported and should not be called.
GetHashCode()	**Obsolete.** This method is not supported and should not be called.
ToHashCode()	Calculates the final hash code after consecutive Add invocations.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

HashCode.Add Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Add<T>(T)	Adds a single value to the hash code.
Add<T>(T, IEqualityComparer<T>)	Adds a single value to the hash code, specifying the type that provides the hash code function.

Add<T>(T)

Adds a single value to the hash code.

C#

```
public void Add<T> (T value);
```

Type Parameters

`T`

The type of the value to add to the hash code.

Parameters

`value` T

The value to add to the hash code.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Add<T>(T, IEqualityComparer<T>)

Adds a single value to the hash code, specifying the type that provides the hash code function.

C#

```
public void Add<T> (T value,
System.Collections.Generic.IEqualityComparer<T>? comparer);
```

Type Parameters

`T`

The type of the value to add to the hash code.

Parameters

`value` T

The value to add to the hash code.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> to use to calculate the hash code. This value can be a null reference (Nothing in Visual Basic), which will use the default equality comparer for `T`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

HashCode.AddBytes(ReadOnlySpan<Byte>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds a span of bytes to the hash code.

C#

```
public void AddBytes (ReadOnlySpan<byte> value);
```

Parameters

`value` ReadOnlySpan<Byte>

The span to add.

Remarks

This method does not guarantee that the result of adding a span of bytes will match the result of adding the same bytes individually.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

HashCode.Combine Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Combine<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Combines eight values into a hash code.
Combine<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Combines seven values into a hash code.
Combine<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Combines six values into a hash code.
Combine<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Combines five values into a hash code.
Combine<T1,T2,T3,T4>(T1, T2, T3, T4)	Combines four values into a hash code.
Combine<T1,T2,T3>(T1, T2, T3)	Combines three values into a hash code.
Combine<T1,T2>(T1, T2)	Combines two values into a hash code.
Combine<T1>(T1)	Diffuses the hash code returned by the specified value.

Combine<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)

Combines eight values into a hash code.

C#

```
public static int Combine<T1,T2,T3,T4,T5,T6,T7,T8> (T1 value1, T2 value2,
T3 value3, T4 value4, T5 value5, T6 value6, T7 value7, T8 value8);
```

Type Parameters

`T1`

The type of the first value to combine into the hash code.

`T2`

The type of the second value to combine into the hash code.

`T3`

The type of the third value to combine into the hash code.

`T4`

The type of the fourth value to combine into the hash code.

`T5`

The type of the fifth value to combine into the hash code.

`T6`

The type of the sixth value to combine into the hash code.

`T7`

The type of the seventh value to combine into the hash code.

`T8`

The type of the eighth value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

`value3` T3

The third value to combine into the hash code.

`value4` T4

The fourth value to combine into the hash code.

`value5` T5

The fifth value to combine into the hash code.

`value6` T6

The sixth value to combine into the hash code.

`value7` T7

The seventh value to combine into the hash code.

`value8` T8

The eighth value to combine into the hash code.

Returns

Int32

The hash code that represents the eight values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)

Combines seven values into a hash code.

C#

```
public static int Combine<T1,T2,T3,T4,T5,T6,T7> (T1 value1, T2 value2, T3 value3, T4 value4, T5 value5, T6 value6, T7 value7);
```

Type Parameters

`T1`

The type of the first value to combine into the hash code.

`T2`

The type of the second value to combine into the hash code.

`T3`

The type of the third value to combine into the hash code.

`T4`

The type of the fourth value to combine into the hash code.

`T5`

The type of the fifth value to combine into the hash code.

`T6`

The type of the sixth value to combine into the hash code.

`T7`

The type of the seventh value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

`value3` T3

The third value to combine into the hash code.

`value4` T4

The fourth value to combine into the hash code.

`value5` T5

The fifth value to combine into the hash code.

`value6` T6

The sixth value to combine into the hash code.

`value7` T7

The seventh value to combine into the hash code.

Returns

Int32

The hash code that represents the seven values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)

Combines six values into a hash code.

C#

```
public static int Combine<T1,T2,T3,T4,T5,T6> (T1 value1, T2 value2, T3
value3, T4 value4, T5 value5, T6 value6);
```

Type Parameters

T1

The type of the first value to combine into the hash code.

T2

The type of the second value to combine into the hash code.

T3

The type of the third value to combine into the hash code.

T4

The type of the fourth value to combine into the hash code.

`T5`

The type of the fifth value to combine into the hash code.

`T6`

The type of the sixth value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

`value3` T3

The third value to combine into the hash code.

`value4` T4

The fourth value to combine into the hash code.

`value5` T5

The fifth value to combine into the hash code.

`value6` T6

The sixth value to combine into the hash code.

Returns

Int32

The hash code that represents the six values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Combine<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)

Combines five values into a hash code.

```
C#

public static int Combine<T1,T2,T3,T4,T5> (T1 value1, T2 value2, T3
value3, T4 value4, T5 value5);
```

Type Parameters

T1

The type of the first value to combine into the hash code.

T2

The type of the second value to combine into the hash code.

T3

The type of the third value to combine into the hash code.

T4

The type of the fourth value to combine into the hash code.

T5

The type of the fifth value to combine into the hash code.

Parameters

value1 T1

The first value to combine into the hash code.

value2 T2

The second value to combine into the hash code.

value3 T3

The third value to combine into the hash code.

`value4` T4

The fourth value to combine into the hash code.

`value5` T5

The fifth value to combine into the hash code.

Returns

Int32

The hash code that represents the five values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1,T2,T3,T4>(T1, T2, T3, T4)

Combines four values into a hash code.

C#

```
public static int Combine<T1,T2,T3,T4> (T1 value1, T2 value2, T3 value3,
T4 value4);
```

Type Parameters

`T1`

The type of the first value to combine into the hash code.

`T2`

The type of the second value to combine into the hash code.

`T3`

The type of the third value to combine into the hash code.

`T4`

The type of the fourth value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

`value3` T3

The third value to combine into the hash code.

`value4` T4

The fourth value to combine into the hash code.

Returns

Int32

The hash code that represents the four values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1,T2,T3>(T1, T2, T3)

Combines three values into a hash code.

C#

```
public static int Combine<T1,T2,T3> (T1 value1, T2 value2, T3 value3);
```

Type Parameters

`T1`

The type of the first value to combine into the hash code.

`T2`

The type of the second value to combine into the hash code.

`T3`

The type of the third value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

`value3` T3

The third value to combine into the hash code.

Returns

Int32

The hash code that represents the three values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1,T2>(T1, T2)

Combines two values into a hash code.

```
C#
public static int Combine<T1,T2> (T1 value1, T2 value2);
```

Type Parameters

`T1`

The type of the first value to combine into the hash code.

`T2`

The type of the second value to combine into the hash code.

Parameters

`value1` T1

The first value to combine into the hash code.

`value2` T2

The second value to combine into the hash code.

Returns

Int32

The hash code that represents the two values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Combine<T1>(T1)

Diffuses the hash code returned by the specified value.

```
C#
```

```
public static int Combine<T1> (T1 value1);
```

Type Parameters

`T1`

The type of the value to add the hash code.

Parameters

`value1` T1

The value to add to the hash code.

Returns

Int32

The hash code that represents the single value.

Remarks

Some data structures assume that hash codes are diffused across a large range. This method improves the quality of the hash code returned by the value and is useful when the underlying data type is simple, for example, an integer value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

HashCode.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> HashCode is a mutable struct and should not be compared with other HashCodes.

This method is not supported and should not be called.

C#

```
[System.Obsolete("HashCode is a mutable struct and should not be compared
with other HashCodes.", true)]
public override bool Equals (object? obj);
```

Parameters

obj Object

Ignored.

Returns

Boolean

This method will always throw a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Always thrown when this method is called.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Standard	*(2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

HashCode.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> HashCode is a mutable struct and should not be compared with other HashCodes. Use ToHashCode to retrieve the computed hash code.

This method is not supported and should not be called.

C#

```
[System.Obsolete("HashCode is a mutable struct and should not be compared with other HashCodes. Use ToHashCode to retrieve the computed hash code.", true)]
public override int GetHashCode ();
```

Returns

Int32

This method will always throw a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Always thrown when this method is called.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*

Product	Versions *(Obsolete)*
.NET Standard	(2.1)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

HashCode.ToHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the final hash code after consecutive Add invocations.

C#

```
public int ToHashCode ();
```

Returns

Int32

The calculated hash code.

Remarks

This method must be called at most once per instance of HashCode.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

HttpStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the HTTP scheme.

C#

```
public class HttpStyleUriParser : UriParser
```

Inheritance Object → UriParser → HttpStyleUriParser

Constructors

Expand table

HttpStyleUriParser()	Create a customizable parser based on the HTTP scheme.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)

IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

HttpStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Create a customizable parser based on the HTTP scheme.

C#

```
public HttpStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncDisposable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a mechanism for releasing unmanaged resources asynchronously.

C#

```
public interface IAsyncDisposable
```

Derived Microsoft.Extensions.DependencyInjection.AsyncServiceScope
System.Collections.Generic.IAsyncEnumerator<T>
System.Data.Common.DbBatch
System.Data.Common.DbCommand
More...

Remarks

For more information about this API, see Supplemental API remarks for IAsyncDisposable.

Methods

Expand table

Dispose Async()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources asynchronously.

Extension Methods

Expand table

ConfigureAwait(IAsyncDisposable, Boolean)	Configures how awaits on the tasks returned from an async disposable are performed.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncDisposable.DisposeAsync Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources asynchronously.

```
C#
public System.Threading.Tasks.ValueTask DisposeAsync ();
```

Returns

ValueTask

A task that represents the asynchronous dispose operation.

Remarks

Use this method to asynchronously close or release unmanaged resources such as files, streams, and handles held by an instance of the class that implements this interface. Using this method instead of IDisposable.Dispose enables you to perform a resource-intensive dispose operation without blocking the main thread of a GUI application for a long time.

> ⚠ **Warning**
>
> If you are using a class that implements the **IAsyncDisposable** interface, you should call its `DisposeAsync` implementation when you are finished using the class. For more information, see the "Using an object that implements IAsyncDisposable" section in the **IAsyncDisposable** topic.

When implementing this method, ensure that all held resources are freed by propagating the call through the containment hierarchy. For example, if an object A allocates an object B, and object B allocates an object C, then A's `DisposeAsync`

implementation must call `DisposeAsync` on B, which must in turn call `DisposeAsync` on C. An object must also call the `DisposeAsync` method of its base class if the base class implements IAsyncDisposable.

If an object's `DisposeAsync` method is called more than once, the object must ignore all calls after the first one and synchronously return a successfully completed ValueTask. The object must not throw an exception if its `DisposeAsync` method is called multiple times. Instance methods other than `DisposeAsync` can throw an ObjectDisposedException when resources are already disposed.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncResult Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the status of an asynchronous operation.

C#

```
public interface IAsyncResult
```

Derived System.IdentityModel.AsyncResult
System.IdentityModel.Selectors.SecurityTokenProvider.SecurityTokenAsyncResult
System.Runtime.Remoting.Messaging.AsyncResult
System.Threading.Tasks.Task
System.Transactions.CommittableTransaction
More...

Examples

The following example demonstrates how to use the AsyncWaitHandle property to get a WaitHandle, and how to wait for an asynchronous call on a delegate. The WaitHandle is signaled when the asynchronous call completes, and you can wait for it by calling the WaitOne method.

The example consists of two classes: the class that contains the method that is called asynchronously, and the class that contains the `Main` method that makes the call.

For more information and more examples of calling methods asynchronously by using delegates, see Calling Synchronous Methods Asynchronously.

C#

```
using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
```

```
{
    public class AsyncDemo
    {
        // The method to be executed asynchronously.
        public string TestMethod(int callDuration, out int threadId)
        {
            Console.WriteLine("Test method begins.");
            Thread.Sleep(callDuration);
            threadId = Thread.CurrentThread.ManagedThreadId;
            return String.Format("My call time was {0}.",
callDuration.ToString());
        }
    }
    // The delegate must have the same signature as the method
    // it will call asynchronously.
    public delegate string AsyncMethodCaller(int callDuration, out int
threadId);
}
```

C#

```
using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class AsyncMain
    {
        static void Main()
        {
            // The asynchronous method puts the thread id here.
            int threadId;

            // Create an instance of the test class.
            AsyncDemo ad = new AsyncDemo();

            // Create the delegate.
            AsyncMethodCaller caller = new AsyncMethodCaller(ad.TestMethod);

            // Initiate the asychronous call.
            IAsyncResult result = caller.BeginInvoke(3000,
                out threadId, null, null);

            Thread.Sleep(0);
            Console.WriteLine("Main thread {0} does some work.",
                Thread.CurrentThread.ManagedThreadId);

            // Wait for the WaitHandle to become signaled.
            result.AsyncWaitHandle.WaitOne();

            // Perform additional processing here.
            // Call EndInvoke to retrieve the results.
            string returnValue = caller.EndInvoke(out threadId, result);
```

```
            // Close the wait handle.
            result.AsyncWaitHandle.Close();

            Console.WriteLine("The call executed on thread {0}, with return
value \"{1}\".",
                threadId, returnValue);
        }
    }
}

/* This example produces output similar to the following:

Main thread 1 does some work.
Test method begins.
The call executed on thread 3, with return value "My call time was 3000.".
 */
```

Remarks

The IAsyncResult interface is implemented by classes containing methods that can operate asynchronously. It is the return type of methods that initiate an asynchronous operation, such as FileStream.BeginRead, and it is passed to methods that conclude an asynchronous operation, such as FileStream.EndRead. IAsyncResult objects are also passed to methods invoked by AsyncCallback delegates when an asynchronous operation completes.

An object that supports the IAsyncResult interface stores state information for an asynchronous operation and provides a synchronization object to allow threads to be signaled when the operation completes.

> **Note**
>
> The **AsyncResult** class is the implementation of **IAsyncResult** that is returned by the `BeginInvoke` method when you use a delegate to call a method asynchronously.

For a detailed description of how the IAsyncResult interface is used, see the Calling Synchronous Methods Asynchronously topic.

Properties

Expand table

AsyncState	Gets a user-defined object that qualifies or contains information about an asynchronous operation.
AsyncWaitHandle	Gets a WaitHandle that is used to wait for an asynchronous operation to complete.
CompletedSynchronously	Gets a value that indicates whether the asynchronous operation completed synchronously.
IsCompleted	Gets a value that indicates whether the asynchronous operation has completed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncResult.AsyncState Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a user-defined object that qualifies or contains information about an asynchronous operation.

```
C#

public object? AsyncState { get; }
```

Property Value

Object

A user-defined object that qualifies or contains information about an asynchronous operation.

Examples

The following code example demonstrates how the AsyncState property is used to pass information to a callback method. The last parameter of the `BeginInvoke` method call is a format string, which the callback method uses to format an output message.

The example consists of two classes: the class that contains the method that is called asynchronously, and the class that contains the `Main` method that makes the call.

For more information about how this callback example works, and more examples of calling methods asynchronously by using delegates, see Calling Synchronous Methods Asynchronously.

```
C#

using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
```

```
    public class AsyncDemo
    {
        // The method to be executed asynchronously.
        public string TestMethod(int callDuration, out int threadId)
        {
            Console.WriteLine("Test method begins.");
            Thread.Sleep(callDuration);
            threadId = Thread.CurrentThread.ManagedThreadId;
            return String.Format("My call time was {0}.",
callDuration.ToString());
        }
    }
    // The delegate must have the same signature as the method
    // it will call asynchronously.
    public delegate string AsyncMethodCaller(int callDuration, out int
threadId);
}
```

C#

```
using System;
using System.Threading;
using System.Runtime.Remoting.Messaging;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class AsyncMain
    {
        static void Main()
        {
            // Create an instance of the test class.
            AsyncDemo ad = new AsyncDemo();

            // Create the delegate.
            AsyncMethodCaller caller = new AsyncMethodCaller(ad.TestMethod);

            // The threadId parameter of TestMethod is an out parameter, so
            // its input value is never used by TestMethod. Therefore, a
dummy
            // variable can be passed to the BeginInvoke call. If the
threadId
            // parameter were a ref parameter, it would have to be a class-
            // level field so that it could be passed to both BeginInvoke
and
            // EndInvoke.
            int dummy = 0;

            // Initiate the asynchronous call, passing three seconds (3000
ms)
            // for the callDuration parameter of TestMethod; a dummy
variable
            // for the out parameter (threadId); the callback delegate; and
            // state information that can be retrieved by the callback
```

```
method.
            // In this case, the state information is a string that can be
used
            // to format a console message.
            IAsyncResult result = caller.BeginInvoke(3000,
                out dummy,
                new AsyncCallback(CallbackMethod),
                "The call executed on thread {0}, with return value \"
{1}\".");

            Console.WriteLine("The main thread {0} continues to execute...",
                Thread.CurrentThread.ManagedThreadId);

            // The callback is made on a ThreadPool thread. ThreadPool
threads
            // are background threads, which do not keep the application
running
            // if the main thread ends. Comment out the next line to
demonstrate
            // this.
            Thread.Sleep(4000);

            Console.WriteLine("The main thread ends.");
        }

        // The callback method must have the same signature as the
        // AsyncCallback delegate.
        static void CallbackMethod(IAsyncResult ar)
        {
            // Retrieve the delegate.
            AsyncResult result = (AsyncResult) ar;
            AsyncMethodCaller caller = (AsyncMethodCaller)
result.AsyncDelegate;

            // Retrieve the format string that was passed as state
            // information.
            string formatString = (string) ar.AsyncState;

            // Define a variable to receive the value of the out parameter.
            // If the parameter were ref rather than out then it would have
to
            // be a class-level field so it could also be passed to
BeginInvoke.
            int threadId = 0;

            // Call EndInvoke to retrieve the results.
            string returnValue = caller.EndInvoke(out threadId, ar);

            // Use the format string to format the output message.
            Console.WriteLine(formatString, threadId, returnValue);
        }
    }
}

/* This example produces output similar to the following:
```

```
The main thread 1 continues to execute...
Test method begins.
The call executed on thread 3, with return value "My call time was 3000.".
The main thread ends.
 */
```

Remarks

This property returns the object that is the last parameter of the method that initiates an asynchronous operation.

Notes to Implementers

Implement this property to allow the caller of an asynchronous operation to obtain an application-defined object specified at the start of the operation.

Notes to Callers

This object can be used to pass state information for the asynchronous operation to an AsyncCallback that you provide.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- AsyncCallback

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IAsyncResult.AsyncWaitHandle Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a WaitHandle that is used to wait for an asynchronous operation to complete.

C#

```
public System.Threading.WaitHandle AsyncWaitHandle { get; }
```

Property Value

WaitHandle

A WaitHandle that is used to wait for an asynchronous operation to complete.

Examples

The following example demonstrates how to use the AsyncWaitHandle property to get a WaitHandle, and how to wait for an asynchronous call on a delegate. The WaitHandle is signaled when the asynchronous call completes, and you can wait for it by calling the WaitOne method.

The example consists of two classes: the class that contains the method that is called asynchronously, and the class that contains the `Main` method that makes the call.

For more information and more examples of calling methods asynchronously by using delegates, see Calling Synchronous Methods Asynchronously.

C#

```
using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class AsyncDemo
    {
        // The method to be executed asynchronously.
```

```
        public string TestMethod(int callDuration, out int threadId)
        {
            Console.WriteLine("Test method begins.");
            Thread.Sleep(callDuration);
            threadId = Thread.CurrentThread.ManagedThreadId;
            return String.Format("My call time was {0}.",
callDuration.ToString());
        }
    }
    // The delegate must have the same signature as the method
    // it will call asynchronously.
    public delegate string AsyncMethodCaller(int callDuration, out int
threadId);
}
```

C#

```
using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class AsyncMain
    {
        static void Main()
        {
            // The asynchronous method puts the thread id here.
            int threadId;

            // Create an instance of the test class.
            AsyncDemo ad = new AsyncDemo();

            // Create the delegate.
            AsyncMethodCaller caller = new AsyncMethodCaller(ad.TestMethod);

            // Initiate the asychronous call.
            IAsyncResult result = caller.BeginInvoke(3000,
                out threadId, null, null);

            Thread.Sleep(0);
            Console.WriteLine("Main thread {0} does some work.",
                Thread.CurrentThread.ManagedThreadId);

            // Wait for the WaitHandle to become signaled.
            result.AsyncWaitHandle.WaitOne();

            // Perform additional processing here.
            // Call EndInvoke to retrieve the results.
            string returnValue = caller.EndInvoke(out threadId, result);

            // Close the wait handle.
            result.AsyncWaitHandle.Close();
```

```
            Console.WriteLine("The call executed on thread {0}, with return
value \"{1}\".",
                threadId, returnValue);
        }
    }
}

/* This example produces output similar to the following:

Main thread 1 does some work.
Test method begins.
The call executed on thread 3, with return value "My call time was 3000.".
 */
```

Remarks

The return value allows the client to wait for an asynchronous operation to complete instead of polling IsCompleted until the operation concludes. The return value can be used to perform a WaitOne, WaitAny, or WaitAll operation.

The common language runtime supplies a number of waitable objects, such as ManualResetEvent, AutoResetEvent, and Mutex, all of which mirror Win32 synchronization primitives.

Notes to Implementers

The object that implements IAsyncResult does not need to create the WaitHandle until the AsyncWaitHandle property is read. It is the choice of the IAsyncResult implementer. However, if the implementer creates AsyncWaitHandle, it is the responsibility of the implementer to signal the WaitHandle that will terminate the wait at the appropriate time. For example, AsyncResult terminates the wait on behalf of the caller when an asynchronously invoked method returns. Once created, AsyncWaitHandle should be kept alive until the user calls the method that concludes the asynchronous operation. At that time the object behind AsyncWaitHandle can be discarded.

Notes to Callers

Clients that wait for the operation to complete (as opposed to polling) use this property to obtain a synchronization object to wait on.

Note: When you use the `BeginInvoke` method of a delegate to call a method asynchronously and obtain a wait handle from the resulting IAsyncResult, we recommend that you close the wait handle as soon as you are finished using it, by

calling the Close() method. If you simply release all references to the wait handle, system resources are freed when garbage collection reclaims the wait handle, but garbage collection works more efficiently when disposable objects are explicitly closed or disposed. For more information, see the AsyncWaitHandle property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncResult.CompletedSynchronously Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the asynchronous operation completed synchronously.

```
C#
public bool CompletedSynchronously { get; }
```

Property Value

Boolean

`true` if the asynchronous operation completed synchronously; otherwise, `false`.

Remarks

If the synchronous completion of the call is detected in the AsyncCallback delegate, it is probable that the thread that initiated the asynchronous operation is the current thread.

Notes to Implementers

Most implementers of the IAsyncResult interface will not use this property and should return `false`. Beginning with the .NET Framework 4.5, a task that is created with the FromAsync(IAsyncResult, Action<IAsyncResult>, TaskCreationOptions) method will not complete if this property is not implemented correctly. See Application Compatibility in 4.5.

Notes to Callers

Use this property to determine if the asynchronous operation completed synchronously. For example, this property can return `true` for an asynchronous I/O operation if the I/O request was small.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IAsyncResult.IsCompleted Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the asynchronous operation has completed.

```
C#

public bool IsCompleted { get; }
```

Property Value

Boolean

`true` if the operation is complete; otherwise, `false`.

Examples

The following example shows how to use the IsCompleted property of the IAsyncResult to discover when an asynchronous call completes. In this case, the asynchronous call is made by using the `BeginInvoke` method of a delegate. You might use the IsCompleted property to poll for completion when making the asynchronous call from a thread that services the user interface. Polling for completion allows the calling thread to continue executing while the asynchronous call executes on a ThreadPool thread.

The example consists of two classes: the class that contains the method that is called asynchronously, and the class that contains the `Main` method that makes the call.

For more information and more examples of calling methods asynchronously by using delegates, see Calling Synchronous Methods Asynchronously.

```
C#

using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
```

```
    public class AsyncDemo
    {
        // The method to be executed asynchronously.
        public string TestMethod(int callDuration, out int threadId)
        {
            Console.WriteLine("Test method begins.");
            Thread.Sleep(callDuration);
            threadId = Thread.CurrentThread.ManagedThreadId;
            return String.Format("My call time was {0}.",
callDuration.ToString());
        }
    }
    // The delegate must have the same signature as the method
    // it will call asynchronously.
    public delegate string AsyncMethodCaller(int callDuration, out int
threadId);
}
```

C#

```
using System;
using System.Threading;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class AsyncMain
    {
        static void Main() {
            // The asynchronous method puts the thread id here.
            int threadId;

            // Create an instance of the test class.
            AsyncDemo ad = new AsyncDemo();

            // Create the delegate.
            AsyncMethodCaller caller = new AsyncMethodCaller(ad.TestMethod);

            // Initiate the asychronous call.
            IAsyncResult result = caller.BeginInvoke(3000,
                out threadId, null, null);

            // Poll while simulating work.
            while(result.IsCompleted == false) {
                Thread.Sleep(250);
                Console.Write(".");
            }

            // Call EndInvoke to retrieve the results.
            string returnValue = caller.EndInvoke(out threadId, result);

            Console.WriteLine("\nThe call executed on thread {0}, with
return value \"{1}\".",
                threadId, returnValue);
```

```
        }
    }
}

/* This example produces output similar to the following:

Test method begins.
.............
The call executed on thread 3, with return value "My call time was 3000.".
 */
```

Remarks

When this property is `true`, you can assume it is safe to discard any resources you allocate for use by the asynchronous operation.

Notes to Implementers

Implementers will typically return the value of a private field or internal test as the value of this property.

Notes to Callers

Clients that poll for operation status (as opposed to waiting on a synchronization object) use this property to determine the status of the operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ICloneable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Supports cloning, which creates a new instance of a class with the same value as an existing instance.

```
C#
public interface ICloneable
```

Derived Microsoft.VisualC.StlClr.Generic.ConstContainerBidirectionalIterator<TValue>
Microsoft.VisualC.StlClr.Generic.ConstContainerRandomAccessIterator<TValue>
Microsoft.VisualC.StlClr.Generic.ConstReverseBidirectionalIterator<TValue>
Microsoft.VisualC.StlClr.Generic.ConstReverseRandomAccessIterator<TValue>
Microsoft.VisualC.StlClr.Generic.ContainerBidirectionalIterator<TValue>
More...

Remarks

The ICloneable interface enables you to provide a customized implementation that creates a copy of an existing object. The ICloneable interface contains one member, the Clone method, which is intended to provide cloning support beyond that supplied by Object.MemberwiseClone. For more information about cloning, deep versus shallow copies, and examples, see the Object.MemberwiseClone method.

Notes to Implementers

The ICloneable interface simply requires that your implementation of the Clone() method return a copy of the current object instance. It does not specify whether the cloning operation performs a deep copy, a shallow copy, or something in between. Nor does it require all property values of the original instance to be copied to the new instance. For example, the Clone() method performs a shallow copy of all properties except the IsReadOnly property; it always sets this property value to `false` in the cloned

object. Because callers of Clone() cannot depend on the method performing a predictable cloning operation, we recommend that ICloneable not be implemented in public APIs.

Methods

Expand table

Clone()	Creates a new object that is a copy of the current instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberwiseClone()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ICloneable.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new object that is a copy of the current instance.

C#

```
public object Clone ();
```

Returns

Object

A new object that is a copy of this instance.

Remarks

The resulting clone must be of the same type as, or compatible with, the original instance.

An implementation of Clone can perform either a deep copy or a shallow copy. In a deep copy, all objects are duplicated; in a shallow copy, only the top-level objects are duplicated and the lower levels contain references. Because callers of Clone cannot depend on the method performing a predictable cloning operation, we recommend that ICloneable not be implemented in public APIs.

See Object.MemberwiseClone for more information on cloning, deep versus shallow copies, and examples.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberwiseClone()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IComparable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a generalized type-specific comparison method that a value type or class implements to order or sort its instances.

C#

```
public interface IComparable
```

Derived System.Boolean
System.Byte
System.Char
System.Data.OracleClient.OracleBinary
System.Data.OracleClient.OracleBoolean
More...

Examples

The following example illustrates the implementation of IComparable and the requisite CompareTo method.

C#

```
using System;
using System.Collections;

public class Temperature : IComparable
{
    // The temperature value
    protected double temperatureF;

    public int CompareTo(object obj) {
        if (obj == null) return 1;

        Temperature otherTemperature = obj as Temperature;
        if (otherTemperature != null)
            return
```

```
this.temperatureF.CompareTo(otherTemperature.temperatureF);
        else
           throw new ArgumentException("Object is not a Temperature");
    }

    public double Fahrenheit
    {
        get
        {
            return this.temperatureF;
        }
        set
        {
            this.temperatureF = value;
        }
    }

    public double Celsius
    {
        get
        {
            return (this.temperatureF - 32) * (5.0/9);
        }
        set
        {
            this.temperatureF = (value * 9.0/5) + 32;
        }
    }
}

public class CompareTemperatures
{
   public static void Main()
   {
      ArrayList temperatures = new ArrayList();
      // Initialize random number generator.
      Random rnd = new Random();

      // Generate 10 temperatures between 0 and 100 randomly.
      for (int ctr = 1; ctr <= 10; ctr++)
      {
         int degrees = rnd.Next(0, 100);
         Temperature temp = new Temperature();
         temp.Fahrenheit = degrees;
         temperatures.Add(temp);
      }

      // Sort ArrayList.
      temperatures.Sort();

      foreach (Temperature temp in temperatures)
         Console.WriteLine(temp.Fahrenheit);
   }
}
// The example displays the following output to the console (individual
```

```
// values may vary because they are randomly generated):
//       2
//       7
//       16
//       17
//       31
//       37
//       58
//       66
//       72
//       95
```

Remarks

This interface is implemented by types whose values can be ordered or sorted. It requires that implementing types define a single method, CompareTo(Object), that indicates whether the position of the current instance in the sort order is before, after, or the same as a second object of the same type. The instance's IComparable implementation is called automatically by methods such as Array.Sort and ArrayList.Sort.

The implementation of the CompareTo(Object) method must return an Int32 that has one of three values, as shown in the following table.

Expand table

Value	Meaning
Less than zero	The current instance precedes the object specified by the CompareTo method in the sort order.
Zero	This current instance occurs in the same position in the sort order as the object specified by the CompareTo method.
Greater than zero	This current instance follows the object specified by the CompareTo method in the sort order.

All numeric types (such as Int32 and Double) implement IComparable, as do String, Char, and DateTime. Custom types should also provide their own implementation of IComparable to enable object instances to be ordered or sorted.

Methods

Expand table

Compare To(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (object? obj);
```

Parameters

obj Object

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `obj` in the sort order.
Zero	This instance occurs in the same position in the sort order as `obj`.
Greater than zero	This instance follows `obj` in the sort order.

Exceptions

ArgumentException

`obj` is not the same type as this instance.

Examples

The following example illustrates the use of CompareTo to compare a `Temperature` object implementing IComparable with another object. The `Temperature` object implements CompareTo by simply wrapping a call to the Int32.CompareTo method.

C#

```
using System;
using System.Collections;

public class Temperature : IComparable
{
    // The temperature value
    protected double temperatureF;

    public int CompareTo(object obj) {
        if (obj == null) return 1;

        Temperature otherTemperature = obj as Temperature;
        if (otherTemperature != null)
            return
this.temperatureF.CompareTo(otherTemperature.temperatureF);
        else
           throw new ArgumentException("Object is not a Temperature");
    }

    public double Fahrenheit
    {
        get
        {
            return this.temperatureF;
        }
        set
        {
            this.temperatureF = value;
        }
    }

    public double Celsius
    {
        get
        {
            return (this.temperatureF - 32) * (5.0/9);
        }
        set
        {
            this.temperatureF = (value * 9.0/5) + 32;
```

```
        }
    }
}

public class CompareTemperatures
{
   public static void Main()
   {
      ArrayList temperatures = new ArrayList();
      // Initialize random number generator.
      Random rnd = new Random();

      // Generate 10 temperatures between 0 and 100 randomly.
      for (int ctr = 1; ctr <= 10; ctr++)
      {
         int degrees = rnd.Next(0, 100);
         Temperature temp = new Temperature();
         temp.Fahrenheit = degrees;
         temperatures.Add(temp);
      }

      // Sort ArrayList.
      temperatures.Sort();

      foreach (Temperature temp in temperatures)
         Console.WriteLine(temp.Fahrenheit);
   }
}
// The example displays the following output to the console (individual
// values may vary because they are randomly generated):
//       2
//       7
//       16
//       17
//       31
//       37
//       58
//       66
//       72
//       95
```

Remarks

The CompareTo method is implemented by types whose values can be ordered or sorted. It is called automatically by methods of non-generic collection objects, such as Array.Sort, to order each member of the array. If a custom class or structure does not implement IComparable, its members cannot be ordered and the sort operation can throw an InvalidOperationException.

This method is only a definition and must be implemented by a specific class or value type to have effect. The meaning of the comparisons specified in the Return Value section ("precedes", "occurs in the same position as", and "follows") depends on the particular implementation.

By definition, any object compares greater than (or follows) `null`, and two null references compare equal to each other.

The parameter, `obj`, must be the same type as the class or value type that implements this interface; otherwise, an ArgumentException is thrown.

Notes to Implementers

For objects A, B and C, the following must be true:

`A.CompareTo(A)` must return zero.

If `A.CompareTo(B)` returns zero, then `B.CompareTo(A)` must return zero.

If `A.CompareTo(B)` returns zero and `B.CompareTo(C)` returns zero, then `A.CompareTo(C)` must return zero.

If `A.CompareTo(B)` returns a value other than zero, then `B.CompareTo(A)` must return a value of the opposite sign.

If `A.CompareTo(B)` returns a value `x` not equal to zero, and `B.CompareTo(C)` returns a value `y` of the same sign as `x`, then `A.CompareTo(C)` must return a value of the same sign as `x` and `y`.

Notes to Callers

Use the CompareTo(Object) method to determine the ordering of instances of a class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IComparable<T> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a generalized comparison method that a value type or class implements to create a type-specific comparison method for ordering or sorting its instances.

C#

```
public interface IComparable<in T>
```

Type Parameters

T

The type of object to compare.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Derived System.Boolean
System.Byte
System.Char
System.DateOnly
System.DateTime
More...

Examples

The following example illustrates the implementation of IComparable<T> for a simple `Temperature` object. The example creates a SortedList<TKey,TValue> collection of strings with `Temperature` object keys, and adds several pairs of temperatures and strings to the list out of sequence. In the call to the Add method, the SortedList<TKey,TValue> collection uses the IComparable<T> implementation to sort the list entries, which are then displayed in order of increasing temperature.

C#

```
using System;
using System.Collections.Generic;

public class Temperature : IComparable<Temperature>
{
    // Implement the generic CompareTo method with the Temperature
    // class as the Type parameter.
    //
    public int CompareTo(Temperature other)
    {
        // If other is not a valid object reference, this instance is
greater.
        if (other == null) return 1;

        // The temperature comparison depends on the comparison of
        // the underlying Double values.
        return m_value.CompareTo(other.m_value);
    }

    // Define the is greater than operator.
    public static bool operator >  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) > 0;
    }

    // Define the is less than operator.
    public static bool operator <  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) < 0;
    }

    // Define the is greater than or equal to operator.
    public static bool operator >=  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) >= 0;
    }

    // Define the is less than or equal to operator.
    public static bool operator <=  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) <= 0;
    }

    // The underlying temperature value.
    protected double m_value = 0.0;

    public double Celsius
    {
```

```
        get
        {
            return m_value - 273.15;
        }
    }

    public double Kelvin
    {
        get
        {
            return m_value;
        }
        set
        {
            if (value < 0.0)
            {
                throw new ArgumentException("Temperature cannot be less than
absolute zero.");
            }
            else
            {
                m_value = value;
            }
        }
    }

    public Temperature(double kelvins)
    {
        this.Kelvin = kelvins;
    }
}

public class Example
{
    public static void Main()
    {
        SortedList<Temperature, string> temps =
            new SortedList<Temperature, string>();

        // Add entries to the sorted list, out of order.
        temps.Add(new Temperature(2017.15), "Boiling point of Lead");
        temps.Add(new Temperature(0), "Absolute zero");
        temps.Add(new Temperature(273.15), "Freezing point of water");
        temps.Add(new Temperature(5100.15), "Boiling point of Carbon");
        temps.Add(new Temperature(373.15), "Boiling point of water");
        temps.Add(new Temperature(600.65), "Melting point of Lead");

        foreach( KeyValuePair<Temperature, string> kvp in temps )
        {
            Console.WriteLine("{0} is {1} degrees Celsius.", kvp.Value,
kvp.Key.Celsius);
        }
    }
}
/* This example displays the following output:
```

```
        Absolute zero is -273.15 degrees Celsius.
        Freezing point of water is 0 degrees Celsius.
        Boiling point of water is 100 degrees Celsius.
        Melting point of Lead is 327.5 degrees Celsius.
        Boiling point of Lead is 1744 degrees Celsius.
        Boiling point of Carbon is 4827 degrees Celsius.
*/
```

Remarks

This interface is implemented by types whose values can be ordered or sorted and provides a strongly typed comparison method for ordering members of a generic collection object. For example, one number can be larger than a second number, and one string can appear in alphabetical order before another. It requires that implementing types define a single method, CompareTo(T), that indicates whether the position of the current instance in the sort order is before, after, or the same as a second object of the same type. Typically, the method is not called directly from developer code. Instead, it is called automatically by methods such as List<T>.Sort() and Add.

Typically, types that provide an IComparable<T> implementation also implement the IEquatable<T> interface. The IEquatable<T> interface defines the Equals method, which determines the equality of instances of the implementing type.

The implementation of the CompareTo(T) method must return an Int32 that has one of three values, as shown in the following table.

Expand table

Value	Meaning
Less than zero	This object precedes the object specified by the CompareTo method in the sort order.
Zero	This current instance occurs in the same position in the sort order as the object specified by the CompareTo method argument.
Greater than zero	This current instance follows the object specified by the CompareTo method argument in the sort order.

All numeric types (such as Int32 and Double) implement IComparable<T>, as do String, Char, and DateTime. Custom types should also provide their own implementation of IComparable<T> to enable object instances to be ordered or sorted.

Notes to Implementers

Replace the type parameter of the IComparable<T> interface with the type that is implementing this interface.

If you implement IComparable<T>, you should overload the `op_GreaterThan`, `op_GreaterThanOrEqual`, `op_LessThan`, and `op_LessThanOrEqual` operators to return values that are consistent with CompareTo(T). In addition, you should also implement IEquatable<T>. See the IEquatable<T> article for complete information.

Methods

Expand table

CompareTo(T)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IComparable
- IEquatable<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IComparable<T>.CompareTo(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (T? other);
```

Parameters

other T

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `other` in the sort order.
Zero	This instance occurs in the same position in the sort order as `other`.
Greater than zero	This instance follows `other` in the sort order.

Examples

The following code example illustrates the implementation of IComparable<T> for a simple `Temperature` object. The example creates a SortedList<TKey,TValue> collection of strings with `Temperature` object keys, and adds several pairs of temperatures and strings to the list out of sequence. In the call to the Add method, the SortedList<TKey,TValue> collection uses the IComparable<T> implementation to sort the list entries, which are then displayed in order of increasing temperature.

C#

```
using System;
using System.Collections.Generic;

public class Temperature : IComparable<Temperature>
{
    // Implement the generic CompareTo method with the Temperature
    // class as the Type parameter.
    //
    public int CompareTo(Temperature other)
    {
        // If other is not a valid object reference, this instance is
greater.
        if (other == null) return 1;

        // The temperature comparison depends on the comparison of
        // the underlying Double values.
        return m_value.CompareTo(other.m_value);
    }

    // Define the is greater than operator.
    public static bool operator >  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) > 0;
    }

    // Define the is less than operator.
    public static bool operator <  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) < 0;
    }

    // Define the is greater than or equal to operator.
    public static bool operator >=  (Temperature operand1, Temperature
operand2)
    {
       return operand1.CompareTo(operand2) >= 0;
    }

    // Define the is less than or equal to operator.
    public static bool operator <=  (Temperature operand1, Temperature
operand2)
```

```
    {
       return operand1.CompareTo(operand2) <= 0;
    }

    // The underlying temperature value.
    protected double m_value = 0.0;

    public double Celsius
    {
        get
        {
            return m_value - 273.15;
        }
    }

    public double Kelvin
    {
        get
        {
            return m_value;
        }
        set
        {
            if (value < 0.0)
            {
                throw new ArgumentException("Temperature cannot be less than
absolute zero.");
            }
            else
            {
                m_value = value;
            }
        }
    }

    public Temperature(double kelvins)
    {
        this.Kelvin = kelvins;
    }
}

public class Example
{
    public static void Main()
    {
        SortedList<Temperature, string> temps =
            new SortedList<Temperature, string>();

        // Add entries to the sorted list, out of order.
        temps.Add(new Temperature(2017.15), "Boiling point of Lead");
        temps.Add(new Temperature(0), "Absolute zero");
        temps.Add(new Temperature(273.15), "Freezing point of water");
        temps.Add(new Temperature(5100.15), "Boiling point of Carbon");
        temps.Add(new Temperature(373.15), "Boiling point of water");
        temps.Add(new Temperature(600.65), "Melting point of Lead");
```

```
        foreach( KeyValuePair<Temperature, string> kvp in temps )
        {
            Console.WriteLine("{0} is {1} degrees Celsius.", kvp.Value,
kvp.Key.Celsius);
        }
    }
}
/* This example displays the following output:
      Absolute zero is -273.15 degrees Celsius.
      Freezing point of water is 0 degrees Celsius.
      Boiling point of water is 100 degrees Celsius.
      Melting point of Lead is 327.5 degrees Celsius.
      Boiling point of Lead is 1744 degrees Celsius.
      Boiling point of Carbon is 4827 degrees Celsius.
*/
```

Remarks

CompareTo provides a strongly typed comparison method for ordering members of a generic collection object. Because of this, it is usually not called directly from developer code. Instead, it is called automatically by methods such as List<T>.Sort() and Add.

This method is only a definition and must be implemented by a specific class or value type to have effect. The meaning of the comparisons specified in the Return Values section ("precedes", "occurs in the same position as", and "follows) depends on the particular implementation.

By definition, any object compares greater than `null`, and two null references compare equal to each other.

Notes to Implementers

For objects A, B, and C, the following must be true:

`A.CompareTo(A)` is required to return zero.

If `A.CompareTo(B)` returns zero, then `B.CompareTo(A)` is required to return zero.

If `A.CompareTo(B)` returns zero and `B.CompareTo(C)` returns zero, then `A.CompareTo(C)` is required to return zero.

If `A.CompareTo(B)` returns a value other than zero, then `B.CompareTo(A)` is required to return a value of the opposite sign.

If `A.CompareTo(B)` returns a value `x` that is not equal to zero, and `B.CompareTo(C)` returns a value `y` of the same sign as `x`, then `A.CompareTo(C)` is required to return a value of the same sign as `x` and `y`.

Notes to Callers

Use the CompareTo(T) method to determine the ordering of instances of a class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Object
- IComparable

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IConvertible Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Defines methods that convert the value of the implementing reference or value type to a common language runtime type that has an equivalent value.

C#

```
[System.CLSCompliant(false)]
public interface IConvertible
```

Derived System.Boolean
System.Byte
System.Char
System.DateTime
System.DBNull
More...

Attributes CLSCompliantAttribute

Examples

The following code sample demonstrates an implementation of IConvertible for a Complex number class, allowing it to be cast first as a Double and then calling the static Convert members on that Double.

C#

```
using System;
```

```
namespace ConsoleApplication2
{

    /// Class that implements IConvertible
    class Complex : IConvertible
    {
        double  x;
        double  y;

        public Complex(double x, double y)
        {
            this.x = x;
            this.y = y;
        }

        public TypeCode GetTypeCode()
        {
            return TypeCode.Object;
        }

        bool IConvertible.ToBoolean(IFormatProvider provider)
        {
            if( (x != 0.0) || (y != 0.0) )
                return true;
            else
                return false;
        }

        double GetDoubleValue()
        {
            return Math.Sqrt(x*x + y*y);
        }

        byte IConvertible.ToByte(IFormatProvider provider)
        {
            return Convert.ToByte(GetDoubleValue());
        }

        char IConvertible.ToChar(IFormatProvider provider)
        {
            return Convert.ToChar(GetDoubleValue());
        }

        DateTime IConvertible.ToDateTime(IFormatProvider provider)
        {
            return Convert.ToDateTime(GetDoubleValue());
        }

        decimal IConvertible.ToDecimal(IFormatProvider provider)
        {
            return Convert.ToDecimal(GetDoubleValue());
        }

        double IConvertible.ToDouble(IFormatProvider provider)
        {
```

```
            return GetDoubleValue();
        }

        short IConvertible.ToInt16(IFormatProvider provider)
        {
            return Convert.ToInt16(GetDoubleValue());
        }

        int IConvertible.ToInt32(IFormatProvider provider)
        {
            return Convert.ToInt32(GetDoubleValue());
        }

        long IConvertible.ToInt64(IFormatProvider provider)
        {
            return Convert.ToInt64(GetDoubleValue());
        }

        sbyte IConvertible.ToSByte(IFormatProvider provider)
        {
            return Convert.ToSByte(GetDoubleValue());
        }

        float IConvertible.ToSingle(IFormatProvider provider)
        {
            return Convert.ToSingle(GetDoubleValue());
        }

        string IConvertible.ToString(IFormatProvider provider)
        {
            return String.Format("({0}, {1})", x, y);
        }

        object IConvertible.ToType(Type conversionType, IFormatProvider
provider)
        {
            return Convert.ChangeType(GetDoubleValue(),conversionType);
        }

        ushort IConvertible.ToUInt16(IFormatProvider provider)
        {
            return Convert.ToUInt16(GetDoubleValue());
        }

        uint IConvertible.ToUInt32(IFormatProvider provider)
        {
            return Convert.ToUInt32(GetDoubleValue());
        }

        ulong IConvertible.ToUInt64(IFormatProvider provider)
        {
            return Convert.ToUInt64(GetDoubleValue());
        }
    }
```

```
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        static void Main(string[] args)
        {

            Complex     testComplex = new Complex(4,7);

            WriteObjectInfo(testComplex);
            WriteObjectInfo(Convert.ToBoolean(testComplex));
            WriteObjectInfo(Convert.ToDecimal(testComplex));
            WriteObjectInfo(Convert.ToString(testComplex));
        }

        static void WriteObjectInfo(object testObject)
        {
            TypeCode    typeCode = Type.GetTypeCode( testObject.GetType() );

            switch( typeCode )
            {
                case TypeCode.Boolean:
                    Console.WriteLine("Boolean: {0}", testObject);
                    break;

                case TypeCode.Double:
                    Console.WriteLine("Double: {0}", testObject);
                    break;

                default:
                    Console.WriteLine("{0}: {1}", typeCode.ToString(),
testObject);
                    break;
            }
        }
    }
}
```

Remarks

This interface provides methods to convert the value of an instance of an implementing type to a common language runtime type that has an equivalent value. The common language runtime types are Boolean, SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double, Decimal, DateTime, Char, and String.

If there is no meaningful conversion to a common language runtime type, then a particular interface method implementation throws InvalidCastException. For example, if this interface is implemented on a Boolean type, the implementation of the ToDateTime

method throws an exception because there is no meaningful DateTime equivalent to a Boolean type.

The common language runtime typically exposes the IConvertible interface through the Convert class. The common language runtime also uses the IConvertible interface internally, in explicit interface implementations, to simplify the code used to support conversions in the Convert class and basic common language runtime types.

In addition to the IConvertible interface, the .NET Framework provides classes called type converters for converting user-defined data types to other data types. For more information, see the Generalized Type Conversion topic.

Notes to Implementers

If you implement the IConvertible interface, your implementation will be called automatically by the ChangeType(Object, Type) method if the Object parameter is an instance of your implementing type and the Type parameter is a common language runtime type.

Most conversion methods have a parameter of type IFormatProvider that represents either the current culture (CurrentCulture) or a specific culture. For the most part, the IConvertible implementations of the base types ignore this parameter. However, you can choose whether to use it in your code.

Methods

Expand table

GetTypeCode()	Returns the TypeCode for this instance.
ToBoolean(IFormatProvider)	Converts the value of this instance to an equivalent Boolean value using the specified culture-specific formatting information.
ToByte(IFormatProvider)	Converts the value of this instance to an equivalent 8-bit unsigned integer using the specified culture-specific formatting information.
ToChar(IFormatProvider)	Converts the value of this instance to an equivalent Unicode character using the specified culture-specific formatting information.
ToDateTime(IFormatProvider)	Converts the value of this instance to an equivalent DateTime using the specified culture-specific formatting information.
ToDecimal(IFormatProvider)	Converts the value of this instance to an equivalent Decimal number using the specified culture-specific formatting information.

ToDouble(IFormatProvider)	Converts the value of this instance to an equivalent double-precision floating-point number using the specified culture-specific formatting information.
ToInt16(IFormatProvider)	Converts the value of this instance to an equivalent 16-bit signed integer using the specified culture-specific formatting information.
ToInt32(IFormatProvider)	Converts the value of this instance to an equivalent 32-bit signed integer using the specified culture-specific formatting information.
ToInt64(IFormatProvider)	Converts the value of this instance to an equivalent 64-bit signed integer using the specified culture-specific formatting information.
ToSByte(IFormatProvider)	Converts the value of this instance to an equivalent 8-bit signed integer using the specified culture-specific formatting information.
ToSingle(IFormatProvider)	Converts the value of this instance to an equivalent single-precision floating-point number using the specified culture-specific formatting information.
ToString(IFormatProvider)	Converts the value of this instance to an equivalent String using the specified culture-specific formatting information.
ToType(Type, IFormatProvider)	Converts the value of this instance to an Object of the specified Type that has an equivalent value, using the specified culture-specific formatting information.
ToUInt16(IFormatProvider)	Converts the value of this instance to an equivalent 16-bit unsigned integer using the specified culture-specific formatting information.
ToUInt32(IFormatProvider)	Converts the value of this instance to an equivalent 32-bit unsigned integer using the specified culture-specific formatting information.
ToUInt64(IFormatProvider)	Converts the value of this instance to an equivalent 64-bit unsigned integer using the specified culture-specific formatting information.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Type Conversion in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IConvertible.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for this instance.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant that is the TypeCode of the class or value type that implements this interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent Boolean value using the specified culture-specific formatting information.

C#

```
public bool ToBoolean (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Boolean

A Boolean value equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 8-bit unsigned integer using the specified culture-specific formatting information.

C#

```
public byte ToByte (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Byte

An 8-bit unsigned integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent Unicode character using the specified culture-specific formatting information.

C#

```
public char ToChar (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Char

A Unicode character equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent DateTime using the specified culture-specific formatting information.

C#

```
public DateTime ToDateTime (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

DateTime

A DateTime instance equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent Decimal number using the specified culture-specific formatting information.

C#

```
public decimal ToDecimal (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Decimal

A Decimal number equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent double-precision floating-point number using the specified culture-specific formatting information.

C#

```
public double ToDouble (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Double

A double-precision floating-point number equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 16-bit signed integer using the specified culture-specific formatting information.

C#

```
public short ToInt16 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Int16

An 16-bit signed integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 32-bit signed integer using the specified culture-specific formatting information.

C#

```
public int ToInt32 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Int32

An 32-bit signed integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 64-bit signed integer using the specified culture-specific formatting information.

C#

```
public long ToInt64 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Int64

An 64-bit signed integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 8-bit signed integer using the specified culture-specific formatting information.

C#

```
public sbyte ToSByte (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

SByte

An 8-bit signed integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent single-precision floating-point number using the specified culture-specific formatting information.

C#

```
public float ToSingle (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Single

A single-precision floating-point number equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToString(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent String using the specified culture-specific formatting information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

String

A String instance equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an Object of the specified Type that has an equivalent value, using the specified culture-specific formatting information.

C#

```
public object ToType (Type conversionType, IFormatProvider? provider);
```

Parameters

`conversionType` Type

The Type to which the value of this instance is converted.

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

Object

An Object instance of type `conversionType` whose value is equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 16-bit unsigned integer using the specified culture-specific formatting information.

C#

```
public ushort ToUInt16 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

UInt16

An 16-bit unsigned integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 32-bit unsigned integer using the specified culture-specific formatting information.

C#

```
public uint ToUInt32 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

UInt32

An 32-bit unsigned integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to an equivalent 64-bit unsigned integer using the specified culture-specific formatting information.

C#

```
public ulong ToUInt64 (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider interface implementation that supplies culture-specific formatting information.

Returns

UInt64

An 64-bit unsigned integer equivalent to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ICustomFormatter Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a method that supports custom formatting of the value of an object.

C#

```
public interface ICustomFormatter
```

Examples

The following example implements ICustomFormatter to allow binary, octal, and hexadecimal formatting of integral values. In this example, a single class, `MyFormatter`, implements both ICustomFormatter and IFormatProvider. Its IFormatProvider.GetFormat method determines whether the `formatType` parameter represents an ICustomFormatter type. If it does, `MyFormatter` returns an instance of itself; otherwise, it returns `null`. Its ICustomFormatter.Format implementation determines whether the format parameter is one of the three supported format strings ("B" for binary, "O" for octal, and "H" for hexadecimal) and formats the `arg` parameter appropriately. Otherwise, if `arg` is not `null`, it calls the `arg` parameter's IFormattable.ToString implementation, if one exists, or its parameterless `ToString` method, if one does not. If `arg` is `null`, the method returns String.Empty.

C#

```
using System;
using System.Globalization;
using System.Numerics;

public class MyFormatter : IFormatProvider, ICustomFormatter
{
    // IFormatProvider.GetFormat implementation.
    public object GetFormat(Type formatType)
    {
        // Determine whether custom formatting object is requested.
        if (formatType == typeof(ICustomFormatter))
            return this;
```

```
        else
            return null;
    }

    // Format number in binary (B), octal (O), or hexadecimal (H).
    public string Format(string format, object arg, IFormatProvider
formatProvider)
    {
        // Handle format string.
        int baseNumber;
        // Handle null or empty format string, string with precision
specifier.
        string thisFmt = String.Empty;
        // Extract first character of format string (precision specifiers
        // are not supported).
        if (!String.IsNullOrEmpty(format))
            thisFmt = format.Length > 1 ? format.Substring(0, 1) : format;

        // Get a byte array representing the numeric value.
        byte[] bytes;
        if (arg is sbyte)
        {
            string byteString = ((sbyte)arg).ToString("X2");
            bytes = new byte[1] { Byte.Parse(byteString,
System.Globalization.NumberStyles.HexNumber) };
        }
        else if (arg is byte)
        {
            bytes = new byte[1] { (byte)arg };
        }
        else if (arg is short)
        {
            bytes = BitConverter.GetBytes((short)arg);
        }
        else if (arg is int)
        {
            bytes = BitConverter.GetBytes((int)arg);
        }
        else if (arg is long)
        {
            bytes = BitConverter.GetBytes((long)arg);
        }
        else if (arg is ushort)
        {
            bytes = BitConverter.GetBytes((ushort)arg);
        }
        else if (arg is uint)
        {
            bytes = BitConverter.GetBytes((uint)arg);
        }
        else if (arg is ulong)
        {
            bytes = BitConverter.GetBytes((ulong)arg);
        }
        else if (arg is BigInteger)
```

```
        {
            bytes = ((BigInteger)arg).ToByteArray();
        }
        else
        {
            try
            {
                return HandleOtherFormats(format, arg);
            }
            catch (FormatException e)
            {
                throw new FormatException(String.Format("The format of '{0}'
is invalid.", format), e);
            }
        }

        switch (thisFmt.ToUpper())
        {
            // Binary formatting.
            case "B":
                baseNumber = 2;
                break;
            case "O":
                baseNumber = 8;
                break;
            case "H":
                baseNumber = 16;
                break;
            // Handle unsupported format strings.
            default:
                try
                {
                    return HandleOtherFormats(format, arg);
                }
                catch (FormatException e)
                {
                    throw new FormatException(String.Format("The format of
'{0}' is invalid.", format), e);
                }
        }

        // Return a formatted string.
        string numericString = String.Empty;
        for (int ctr = bytes.GetUpperBound(0); ctr >=
bytes.GetLowerBound(0); ctr--)
        {
            string byteString = Convert.ToString(bytes[ctr], baseNumber);
            if (baseNumber == 2)
                byteString = new String('0', 8 - byteString.Length) +
byteString;
            else if (baseNumber == 8)
                byteString = new String('0', 4 - byteString.Length) +
byteString;
            // Base is 16.
            else
```

```
                byteString = new String('0', 2 - byteString.Length) +
byteString;

            numericString += byteString + " ";
        }
        return numericString.Trim();
    }

    private string HandleOtherFormats(string format, object arg)
    {
        if (arg is IFormattable)
            return ((IFormattable)arg).ToString(format,
CultureInfo.CurrentCulture);
        else if (arg != null)
            return arg.ToString();
        else
            return String.Empty;
    }
}
```

`MyFormatter` can then be used to provide custom formatting by passing a `MyFormatter` object as the `provider` parameter of the Format method, as the following example shows.

C#

```
public class Example
{
    public static void Main()
    {
        Console.WindowWidth = 100;

        byte byteValue = 124;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0} (binary: {0:B}) (hex: {0:H})",
byteValue));

        int intValue = 23045;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0} (binary: {0:B}) (hex: {0:H})",
intValue));

        ulong ulngValue = 31906574882;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0}\n   (binary: {0:B})\n   (hex:
{0:H})",
                                        ulngValue));

        BigInteger bigIntValue = BigInteger.Multiply(Int64.MaxValue, 2);
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0}\n   (binary: {0:B})\n   (hex:
{0:H})",
```

```
                                        bigIntValue));
    }
}
// The example displays the following output:
//    124 (binary: 01111100) (hex: 7c)
//    23045 (binary: 00000000 00000000 01011010 00000101) (hex: 00 00 5a 05)
//    31906574882
//       (binary: 00000000 00000000 00000000 00000111 01101101 11000111
10110010 00100010)
//       (hex: 00 00 00 07 6d c7 b2 22)
//    18446744073709551614
//       (binary: 00000000 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111110)
//       (hex: 00 ff ff ff ff ff ff ff fe)
```

Remarks

The ICustomFormatter interface includes a single method, ICustomFormatter.Format. When this interface is implemented by a reference or value type, the Format method returns a custom-formatted string representation of an object's value.

Typically, the ICustomFormatter interface is implemented with the IFormatProvider interface to customize the behavior of two .NET Framework composite string formatting methods that include an IFormatProvider parameter. Specifically, the ICustomFormatter interface can provide custom formatting of the value of an object passed to the String.Format(IFormatProvider, String, Object[]) and StringBuilder.AppendFormat(IFormatProvider, String, Object[]) methods.

Providing a custom representation of an object's value requires that you do the following:

1. Define a class that implements the ICustomFormatter interface and its single member, the Format method.

2. Define a class that implements the IFormatProvider interface and its single member, the GetFormat method. The GetFormat method returns an instance of your ICustomFormatter implementation. Often, a single class implements both ICustomFormatter and IFormatProvider. In that case, the class's `GetFormat` implementation just returns an instance of itself.

3. Pass the IFormatProvider implementation as the `provider` argument of the String.Format(IFormatProvider, String, Object[]) method or a comparable method.

The .NET library method will then use your custom formatting instead of its own.

Notes to Implementers

The common language runtime attempts to use your ICustomFormatter implementation for every format item in a composite format string. As a result, you should expect that your ICustomFormatter implementation will be called to provide formatting services to objects or values that it is not designed to handle. In these cases, your Format(String, Object, IFormatProvider) method must call the appropriate formatting method for that object or value.

There are two kinds of ICustomFormatter implementations: intrinsic and extension.

Intrinsic implementations are implementations that provide custom formatting for an application-defined object. In this case, your implementation should include the following:

- A definition of format strings that define the formatting of the object. Format strings are optional. Typically, a "G" or "g" format string defines the general (or most commonly used) format. However, you are free to define any format strings that you choose. You are also free to decide whether they are case-sensitive or case-insensitive.

- A test to ensure that the type of the object passed to your Format(String, Object, IFormatProvider) method is your application-defined type. If it is not, you should call the object's IFormattable implementation, if one exists, or its ToString() method, if it does not. You should be prepared to handle any exceptions these method calls might throw.

- Code to handle a null format string, if your implementation supports format strings. The most common approach is to replace a null format string with the general format specifier.

- Code to handle any format strings that your implementation supports.

- Code to handle format strings that you do not support. The most common approach is to throw a FormatException, although you can provide default formatting.

Extension implementations are implementations that provide custom formatting for a type that already has formatting support. For example, you could define a `CustomerNumberFormatter` that formats an integral type with hyphens between specific digits. In this case, your implementation should include the following:

- A definition of format strings that extend the formatting of the object. These format strings are required, but they must not conflict with the type's existing

format strings. For example, if you are extending formatting for the Int32 type, you should not implement the "C", "D", "E", "F", and "G" format specifiers, among others.

- A test that the type of the object passed to your Format(String, Object, IFormatProvider) method is a type whose formatting your extension supports. If it is not, call the object's IFormattable implementation, if one exists, or the object's parameterless ToString() method, if it does not. You should be prepared to handle any exceptions these method calls might throw.
- Code to handle any format strings that your extension supports.
- Code to handle any format strings that your extension does not support. These should be passed on to the type's IFormattable implementation. You should be prepared to handle any exceptions these method calls might throw.

Methods

Expand table

Format(String, Object, IFormatProvider)	Converts the value of a specified object to an equivalent string representation using specified format and culture-specific formatting information.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IFormatProvider

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ICustomFormatter.Format(String, Object, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of a specified object to an equivalent string representation using specified format and culture-specific formatting information.

C#

```
public string Format (string? format, object? arg, IFormatProvider?
formatProvider);
```

Parameters

`format` String

A format string containing formatting specifications.

`arg` Object

An object to format.

`formatProvider` IFormatProvider

An object that supplies format information about the current instance.

Returns

String

The string representation of the value of `arg`, formatted as specified by `format` and `formatProvider`.

Examples

The following example implements ICustomFormatter to allow binary, octal, and hexadecimal formatting of integral values. Its ICustomFormatter.Format implementation

determines whether the format parameter is one of the three supported format strings ("B" for binary, "O" for octal, and "H" for hexadecimal) and formats the `arg` parameter appropriately. Otherwise, if `arg` is not `null`, it calls the `arg` parameter's IFormattable.ToString implementation, if one exists, or its parameterless `ToString` method, if one does not. If `arg` is `null`, the method returns String.Empty.

C#

```
using System;
using System.Globalization;
using System.Numerics;

public class MyFormatter : IFormatProvider, ICustomFormatter
{
    // IFormatProvider.GetFormat implementation.
    public object GetFormat(Type formatType)
    {
        // Determine whether custom formatting object is requested.
        if (formatType == typeof(ICustomFormatter))
            return this;
        else
            return null;
    }

    // Format number in binary (B), octal (O), or hexadecimal (H).
    public string Format(string format, object arg, IFormatProvider
formatProvider)
    {
        // Handle format string.
        int baseNumber;
        // Handle null or empty format string, string with precision
specifier.
        string thisFmt = String.Empty;
        // Extract first character of format string (precision specifiers
        // are not supported).
        if (!String.IsNullOrEmpty(format))
            thisFmt = format.Length > 1 ? format.Substring(0, 1) : format;

        // Get a byte array representing the numeric value.
        byte[] bytes;
        if (arg is sbyte)
        {
            string byteString = ((sbyte)arg).ToString("X2");
            bytes = new byte[1] { Byte.Parse(byteString,
System.Globalization.NumberStyles.HexNumber) };
        }
        else if (arg is byte)
        {
            bytes = new byte[1] { (byte)arg };
        }
        else if (arg is short)
        {
```

```
            bytes = BitConverter.GetBytes((short)arg);
        }
        else if (arg is int)
        {
            bytes = BitConverter.GetBytes((int)arg);
        }
        else if (arg is long)
        {
            bytes = BitConverter.GetBytes((long)arg);
        }
        else if (arg is ushort)
        {
            bytes = BitConverter.GetBytes((ushort)arg);
        }
        else if (arg is uint)
        {
            bytes = BitConverter.GetBytes((uint)arg);
        }
        else if (arg is ulong)
        {
            bytes = BitConverter.GetBytes((ulong)arg);
        }
        else if (arg is BigInteger)
        {
            bytes = ((BigInteger)arg).ToByteArray();
        }
        else
        {
            try
            {
                return HandleOtherFormats(format, arg);
            }
            catch (FormatException e)
            {
                throw new FormatException(String.Format("The format of '{0}'
is invalid.", format), e);
            }
        }

        switch (thisFmt.ToUpper())
        {
            // Binary formatting.
            case "B":
                baseNumber = 2;
                break;
            case "O":
                baseNumber = 8;
                break;
            case "H":
                baseNumber = 16;
                break;
            // Handle unsupported format strings.
            default:
                try
                {
```

```
                    return HandleOtherFormats(format, arg);
                }
                catch (FormatException e)
                {
                    throw new FormatException(String.Format("The format of
'{0}' is invalid.", format), e);
                }
        }

        // Return a formatted string.
        string numericString = String.Empty;
        for (int ctr = bytes.GetUpperBound(0); ctr >=
bytes.GetLowerBound(0); ctr--)
        {
            string byteString = Convert.ToString(bytes[ctr], baseNumber);
            if (baseNumber == 2)
                byteString = new String('0', 8 - byteString.Length) +
byteString;
            else if (baseNumber == 8)
                byteString = new String('0', 4 - byteString.Length) +
byteString;
            // Base is 16.
            else
                byteString = new String('0', 2 - byteString.Length) +
byteString;

            numericString += byteString + " ";
        }
        return numericString.Trim();
    }

    private string HandleOtherFormats(string format, object arg)
    {
        if (arg is IFormattable)
            return ((IFormattable)arg).ToString(format,
CultureInfo.CurrentCulture);
        else if (arg != null)
            return arg.ToString();
        else
            return String.Empty;
    }
}
```

`MyFormatter` can then be used to provide custom formatting by passing a `MyFormatter` object as the `provider` parameter of the Format method, as the following example shows.

C#

```
public class Example
{
    public static void Main()
```

```
    {
        Console.WindowWidth = 100;

        byte byteValue = 124;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0} (binary: {0:B}) (hex: {0:H})",
byteValue));

        int intValue = 23045;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0} (binary: {0:B}) (hex: {0:H})",
intValue));

        ulong ulngValue = 31906574882;
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0}\n   (binary: {0:B})\n   (hex:
{0:H})",
                                        ulngValue));

        BigInteger bigIntValue = BigInteger.Multiply(Int64.MaxValue, 2);
        Console.WriteLine(String.Format(new MyFormatter(),
                                        "{0}\n   (binary: {0:B})\n   (hex:
{0:H})",
                                        bigIntValue));
    }
}
// The example displays the following output:
//    124 (binary: 01111100) (hex: 7c)
//    23045 (binary: 00000000 00000000 01011010 00000101) (hex: 00 00 5a 05)
//    31906574882
//       (binary: 00000000 00000000 00000000 00000111 01101101 11000111
10110010 00100010)
//       (hex: 00 00 00 07 6d c7 b2 22)
//    18446744073709551614
//       (binary: 00000000 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111110)
//       (hex: 00 ff ff ff ff ff ff ff fe)
```

Remarks

ICustomFormatter.Format is a callback method. It is called by a method that supports custom formatting, such as String.Format(IFormatProvider, String, Object[]) or StringBuilder.AppendFormat(IFormatProvider, String, Object[]). The implementation is called once for each format item in a composite format string. For example, in the following statement, the ICustomFormatter.Format method is called three times.

C#

```
Console.WriteLine(String.Format(new MyFormatter(),
                                "{0} (binary: {0:B}) (hex: {0:H})",
```

```
byteValue));
```

The `arg` parameter is the object in the object list whose zero-based position corresponds to the index of a particular format item.

The `format` parameter contains a format string, which is the `formatString` component of a format item. If the format item has no `formatString` component, the value of `format` is `null`. If `format` is `null`, depending on the type of `arg`, you may be able to use the default format specification of your choice.

The `formatProvider` parameter is the IFormatProvider implementation that provides formatting for `arg`. Typically, it is an instance of your ICustomFormatter implementation. If `formatProvider` is `null`, ignore that parameter.

Your implementation of the Format method must include the following functionality so the .NET Framework can provide formatting you do not support. If your format method does not support a format, determine whether the object being formatted implements the IFormattable interface. If it does, invoke the IFormattable.ToString method of that interface. Otherwise, invoke the default Object.ToString method of the underlying object. The following code illustrates this pattern.

C#

```
if (arg is IFormattable)
    return ((IFormattable)arg).ToString(format, CultureInfo.CurrentCulture);
else if (arg != null)
    return arg.ToString();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IFormattable

- Formatting Types in .NET
- Composite Formatting

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IDisposable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a mechanism for releasing unmanaged resources.

C#

```
public interface IDisposable
```

Derived Microsoft.Extensions.Caching.Memory.ICacheEntry
Microsoft.Extensions.Caching.Memory.IMemoryCache
Microsoft.Extensions.Caching.Memory.MemoryCache
Microsoft.Extensions.Caching.Redis.RedisCache
Microsoft.Extensions.Caching.StackExchangeRedis.RedisCache
More...

Examples

The following example demonstrates how to create a resource class that implements the IDisposable interface.

C#

```
using System;
using System.ComponentModel;

// The following example demonstrates how to create
// a resource class that implements the IDisposable interface
// and the IDisposable.Dispose method.

public class DisposeExample
{
    // A base class that implements IDisposable.
    // By implementing IDisposable, you are announcing that
    // instances of this type allocate scarce resources.
    public class MyResource: IDisposable
    {
        // Pointer to an external unmanaged resource.
```

```
        private IntPtr handle;
        // Other managed resource this class uses.
        private Component component = new Component();
        // Track whether Dispose has been called.
        private bool disposed = false;

        // The class constructor.
        public MyResource(IntPtr handle)
        {
            this.handle = handle;
        }

        // Implement IDisposable.
        // Do not make this method virtual.
        // A derived class should not be able to override this method.
        public void Dispose()
        {
            Dispose(disposing: true);
            // This object will be cleaned up by the Dispose method.
            // Therefore, you should call GC.SuppressFinalize to
            // take this object off the finalization queue
            // and prevent finalization code for this object
            // from executing a second time.
            GC.SuppressFinalize(this);
        }

        // Dispose(bool disposing) executes in two distinct scenarios.
        // If disposing equals true, the method has been called directly
        // or indirectly by a user's code. Managed and unmanaged resources
        // can be disposed.
        // If disposing equals false, the method has been called by the
        // runtime from inside the finalizer and you should not reference
        // other objects. Only unmanaged resources can be disposed.
        protected virtual void Dispose(bool disposing)
        {
            // Check to see if Dispose has already been called.
            if(!this.disposed)
            {
                // If disposing equals true, dispose all managed
                // and unmanaged resources.
                if(disposing)
                {
                    // Dispose managed resources.
                    component.Dispose();
                }

                // Call the appropriate methods to clean up
                // unmanaged resources here.
                // If disposing is false,
                // only the following code is executed.
                CloseHandle(handle);
                handle = IntPtr.Zero;

                // Note disposing has been done.
                disposed = true;
```

```
            }
        }

        // Use interop to call the method necessary
        // to clean up the unmanaged resource.
        [System.Runtime.InteropServices.DllImport("Kernel32")]
        private extern static Boolean CloseHandle(IntPtr handle);

        // Use C# finalizer syntax for finalization code.
        // This finalizer will run only if the Dispose method
        // does not get called.
        // It gives your base class the opportunity to finalize.
        // Do not provide finalizer in types derived from this class.
        ~MyResource()
        {
            // Do not re-create Dispose clean-up code here.
            // Calling Dispose(disposing: false) is optimal in terms of
            // readability and maintainability.
            Dispose(disposing: false);
        }
    }
    public static void Main()
    {
        // Insert code here to create
        // and use the MyResource object.
    }
}
```

Remarks

For more information about this API, see Supplemental API remarks for IDisposable.

Methods

Expand table

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- SafeFileHandle
- Implementing a Dispose Method

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IDisposable.Dispose Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

C#

```
public void Dispose ();
```

Examples

The following example shows how you can implement the Dispose method.

C#

```
using System;
using System.ComponentModel;

// The following example demonstrates how to create
// a resource class that implements the IDisposable interface
// and the IDisposable.Dispose method.

public class DisposeExample
{
    // A base class that implements IDisposable.
    // By implementing IDisposable, you are announcing that
    // instances of this type allocate scarce resources.
    public class MyResource: IDisposable
    {
        // Pointer to an external unmanaged resource.
        private IntPtr handle;
        // Other managed resource this class uses.
        private Component component = new Component();
        // Track whether Dispose has been called.
        private bool disposed = false;

        // The class constructor.
        public MyResource(IntPtr handle)
        {
            this.handle = handle;
```

```
        }

        // Implement IDisposable.
        // Do not make this method virtual.
        // A derived class should not be able to override this method.
        public void Dispose()
        {
            Dispose(disposing: true);
            // This object will be cleaned up by the Dispose method.
            // Therefore, you should call GC.SuppressFinalize to
            // take this object off the finalization queue
            // and prevent finalization code for this object
            // from executing a second time.
            GC.SuppressFinalize(this);
        }

        // Dispose(bool disposing) executes in two distinct scenarios.
        // If disposing equals true, the method has been called directly
        // or indirectly by a user's code. Managed and unmanaged resources
        // can be disposed.
        // If disposing equals false, the method has been called by the
        // runtime from inside the finalizer and you should not reference
        // other objects. Only unmanaged resources can be disposed.
        protected virtual void Dispose(bool disposing)
        {
            // Check to see if Dispose has already been called.
            if(!this.disposed)
            {
                // If disposing equals true, dispose all managed
                // and unmanaged resources.
                if(disposing)
                {
                    // Dispose managed resources.
                    component.Dispose();
                }

                // Call the appropriate methods to clean up
                // unmanaged resources here.
                // If disposing is false,
                // only the following code is executed.
                CloseHandle(handle);
                handle = IntPtr.Zero;

                // Note disposing has been done.
                disposed = true;
            }
        }

        // Use interop to call the method necessary
        // to clean up the unmanaged resource.
        [System.Runtime.InteropServices.DllImport("Kernel32")]
        private extern static Boolean CloseHandle(IntPtr handle);

        // Use C# finalizer syntax for finalization code.
        // This finalizer will run only if the Dispose method
```

```
        // does not get called.
        // It gives your base class the opportunity to finalize.
        // Do not provide finalizer in types derived from this class.
        ~MyResource()
        {
            // Do not re-create Dispose clean-up code here.
            // Calling Dispose(disposing: false) is optimal in terms of
            // readability and maintainability.
            Dispose(disposing: false);
        }
    }
    public static void Main()
    {
        // Insert code here to create
        // and use the MyResource object.
    }
}
```

Remarks

Use this method to close or release unmanaged resources such as files, streams, and handles held by an instance of the class that implements this interface. By convention, this method is used for all tasks associated with freeing resources held by an object, or preparing an object for reuse.

> ⚠ **Warning**
>
> If you are using a class that implements the **IDisposable** interface, you should call its **Dispose** implementation when you are finished using the class. For more information, see the "Using an object that implements IDisposable" section in the **IDisposable** topic.

When implementing this method, ensure that all held resources are freed by propagating the call through the containment hierarchy. For example, if an object A allocates an object B, and object B allocates an object C, then A's Dispose implementation must call Dispose on B, which must in turn call Dispose on C.

> ⓘ **Important**
>
> The C++ compiler supports deterministic disposal of resources and does not allow direct implementation of the **Dispose** method.

An object must also call the Dispose method of its base class if the base class implements IDisposable. For more information about implementing IDisposable on a base class and its subclasses, see the "IDisposable and the inheritance hierarchy" section in the IDisposable topic.

If an object's Dispose method is called more than once, the object must ignore all calls after the first one. The object must not throw an exception if its Dispose method is called multiple times. Instance methods other than Dispose can throw an ObjectDisposedException when resources are already disposed.

Users might expect a resource type to use a particular convention to denote an allocated state versus a freed state. An example of this is stream classes, which are traditionally thought of as open or closed. The implementer of a class that has such a convention might choose to implement a public method with a customized name, such as `Close`, that calls the Dispose method.

Because the Dispose method must be called explicitly, there is always a danger that the unmanaged resources will not be released, because the consumer of an object fails to call its Dispose method. There are two ways to avoid this:

- Wrap the managed resource in an object derived from System.Runtime.InteropServices.SafeHandle. Your Dispose implementation then calls the Dispose method of the System.Runtime.InteropServices.SafeHandle instances. For more information, see "The SafeHandle alternative" section in the Object.Finalize topic.

- Implement a finalizer to free resources when Dispose is not called. By default, the garbage collector automatically calls an object's finalizer before reclaiming its memory. However, if the Dispose method has been called, it is typically unnecessary for the garbage collector to call the disposed object's finalizer. To prevent automatic finalization, Dispose implementations can call the GC.SuppressFinalize method.

When you use an object that accesses unmanaged resources, such as a StreamWriter, a good practice is to create the instance with a `using` statement. The `using` statement automatically closes the stream and calls Dispose on the object when the code that is using it has completed. For an example, see the StreamWriter class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Implementing a Dispose Method

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IEquatable<T> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a generalized method that a value type or class implements to create a type-specific method for determining equality of instances.

```
C#
public interface IEquatable<T>
```

Type Parameters

T

The type of objects to compare.

Derived Microsoft.Extensions.AsyncState.AsyncStateToken
Microsoft.Extensions.Compliance.Classification.DataClassification
Microsoft.Extensions.Compliance.Classification.DataClassificationSet
Microsoft.Extensions.Compliance.Testing.RedactedData
Microsoft.Extensions.Compliance.Testing.RedactorRequested
More...

Examples

See the example for the IEquatable<T>.Equals method.

Remarks

This interface is implemented by types whose values can be equated (for example, the numeric and string classes). A value type or class implements the Equals method to create a type-specific method suitable for determining equality of instances.

Note

The **IComparable<T>** interface defines the **CompareTo** method, which determines the sort order of instances of the implementing type. The **IEquatable<T>** interface defines the **Equals** method, which determines the equality of instances of the implementing type.

The IEquatable<T> interface is used by generic collection objects such as Dictionary<TKey,TValue>, List<T>, and LinkedList<T> when testing for equality in such methods as `Contains`, `IndexOf`, `LastIndexOf`, and `Remove`. It should be implemented for any object that might be stored in a generic collection.

Notes to Implementers

Replace the type parameter of the IEquatable<T> interface with the type that is implementing this interface.

If you implement IEquatable<T>, you should also override the base class implementations of Equals(Object) and GetHashCode() so that their behavior is consistent with that of the Equals(T) method. If you do override Equals(Object), your overridden implementation is also called in calls to the static `Equals(System.Object, System.Object)` method on your class. In addition, you should overload the `op_Equality` and `op_Inequality` operators. This ensures that all tests for equality return consistent results.

For information on overriding Equals(Object), see Equals(Object).

For a value type, you should always implement IEquatable<T> and override Equals(Object) for better performance. Equals(Object) boxes value types and relies on reflection to compare two values for equality. Both your implementation of Equals(T) and your override of Equals(Object) should return consistent results.

If you implement IEquatable<T>, you should also implement IComparable<T> if instances of your type can be ordered or sorted. If your type implements IComparable<T>, you almost always also implement IEquatable<T>.

Note that there are some designs where a type supports an order relation, but equality may be distinct from an ordering relation. Consider a `Person` class where you sort alphabetically. Two people with the same name sort the same, but are not the same person.

Methods

Expand table

Equals(T)	Indicates whether the current object is equal to another object of the same type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IComparable<T>
- IComparable

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IEquatable<T>.Equals(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current object is equal to another object of the same type.

C#

```
public bool Equals (T? other);
```

Parameters

other T

An object to compare with this object.

Returns

Boolean

`true` if the current object is equal to the `other` parameter; otherwise, `false`.

Examples

The following example shows the partial implementation of a `Person` class that implements IEquatable<T> and has two properties, `LastName` and `SSN`. The Equals method returns `True` if the `SSN` property of two `Person` objects is identical; otherwise, it returns `False`.

C#

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class Person : IEquatable<Person>
{
   private string uniqueSsn;
   private string lName;
```

```
   public Person(string lastName, string ssn)
   {
      if (Regex.IsMatch(ssn, @"\d{9}"))
        uniqueSsn = $"{ssn.Substring(0, 3)}-{ssn.Substring(3, 2)}-
{ssn.Substring(5, 4)}";
      else if (Regex.IsMatch(ssn, @"\d{3}-\d{2}-\d{4}"))
         uniqueSsn = ssn;
      else
         throw new FormatException("The social security number has an
invalid format.");

      this.LastName = lastName;
   }

   public string SSN
   {
      get { return this.uniqueSsn; }
   }

   public string LastName
   {
      get { return this.lName; }
      set {
         if (String.IsNullOrEmpty(value))
            throw new ArgumentException("The last name cannot be null or
empty.");
         else
            this.lName = value;
      }
   }

   public bool Equals(Person other)
   {
      if (other == null)
         return false;

      if (this.uniqueSsn == other.uniqueSsn)
         return true;
      else
         return false;
   }

   public override bool Equals(Object obj)
   {
      if (obj == null)
         return false;

      Person personObj = obj as Person;
      if (personObj == null)
         return false;
      else
         return Equals(personObj);
   }
```

```
    public override int GetHashCode()
    {
       return this.SSN.GetHashCode();
    }

    public static bool operator == (Person person1, Person person2)
    {
       if (((object)person1) == null || ((object)person2) == null)
          return Object.Equals(person1, person2);

       return person1.Equals(person2);
    }

    public static bool operator != (Person person1, Person person2)
    {
       if (((object)person1) == null || ((object)person2) == null)
          return ! Object.Equals(person1, person2);

       return ! (person1.Equals(person2));
    }
 }
```

`Person` objects can then be stored in a List<T> object and can be identified by the `Contains` method, as the following example shows.

C#

```
public class TestIEquatable
{
   public static void Main()
   {
      // Create a Person object for each job applicant.
      Person applicant1 = new Person("Jones", "099-29-4999");
      Person applicant2 = new Person("Jones", "199-29-3999");
      Person applicant3 = new Person("Jones", "299-49-6999");

      // Add applicants to a List object.
      List<Person> applicants = new List<Person>();
      applicants.Add(applicant1);
      applicants.Add(applicant2);
      applicants.Add(applicant3);

       // Create a Person object for the final candidate.
       Person candidate = new Person("Jones", "199-29-3999");
       if (applicants.Contains(candidate))
          Console.WriteLine("Found {0} (SSN {1}).",
                            candidate.LastName, candidate.SSN);
      else
         Console.WriteLine("Applicant {0} not found.", candidate.SSN);

      // Call the shared inherited Equals(Object, Object) method.
      // It will in turn call the IEquatable(Of T).Equals implementation.
      Console.WriteLine("{0}({1}) already on file: {2}.",
```

```
                        applicant2.LastName,
                        applicant2.SSN,
                        Person.Equals(applicant2, candidate));
    }
}
// The example displays the following output:
//       Found Jones (SSN 199-29-3999).
//       Jones(199-29-3999) already on file: True.
```

Remarks

The implementation of the Equals method is intended to perform a test for equality with another object of type `T`, the same type as the current object. The Equals(T) method is called in the following circumstances:

- When the `Equals` method is called and the `other` argument is a strongly-typed object of type `T`. (If `other` is not of type `T`, the base Object.Equals(Object) method is called. Of the two methods, IEquatable<T>.Equals offers slightly better performance.)
- When the search methods of a number of generic collection objects are called. Some of these types and their methods include the following:
  - Some of the generic overloads of the BinarySearch method.
  - The search methods of the List<T> class, including List<T>.Contains(T), List<T>.IndexOf, List<T>.LastIndexOf, and List<T>.Remove.
  - The search methods of the Dictionary<TKey,TValue> class, including ContainsKey and Remove.
  - The search methods of the generic LinkedList<T> class, including LinkedList<T>.Contains and Remove.

In other words, to handle the possibility that objects of a class will be stored in an array or a generic collection object, it is a good idea to implement IEquatable<T> so that the object can be easily identified and manipulated.

When implementing the Equals method, define equality appropriately for the type specified by the generic type argument. For example, if the type argument is Int32, define equality appropriately for the comparison of two 32-bit signed integers.

Notes to Implementers

If you implement Equals(T), you should also override the base class implementations of Equals(Object) and GetHashCode() so that their behavior is consistent with that of the Equals(T) method. If you do override Equals(Object), your overridden implementation is also called in calls to the static `Equals(System.Object, System.Object)` method on your class. In addition, you should overload the `op_Equality` and `op_Inequality` operators. This ensures that all tests for equality return consistent results, which the example illustrates.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IFormatProvider Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a mechanism for retrieving an object to control formatting.

C#

```
public interface IFormatProvider
```

Derived System.Globalization.CultureInfo
System.Globalization.DateTimeFormatInfo
System.Globalization.NumberFormatInfo

Examples

The following example illustrates how an IFormatProvider implementation can change the representation of a date and time value. In this case, a single date is displayed by using CultureInfo objects that represent four different cultures.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      DateTime dateValue = new DateTime(2009, 6, 1, 16, 37, 0);
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("fr-FR"),
                                 new CultureInfo("it-IT"),
                                 new CultureInfo("de-DE") };
      foreach (CultureInfo culture in cultures)
         Console.WriteLine("{0}: {1}", culture.Name,
dateValue.ToString(culture));
   }
}
// The example displays the following output:
```

```
//       en-US: 6/1/2009 4:37:00 PM
//       fr-FR: 01/06/2009 16:37:00
//       it-IT: 01/06/2009 16.37.00
//       de-DE: 01.06.2009 16:37:00
```

The following example illustrates the use of a class that implements the IFormatProvider interface and the GetFormat method. The `AcctNumberFormat` class converts an Int64 value that represents an account number to a formatted 12-digit account number. Its `GetFormat` method returns a reference to the current `AcctNumberFormat` instance if the `formatType` parameter refers to a class that implements ICustomFormatter; otherwise, `GetFormat` returns `null`.

C#

```
public class AcctNumberFormat : IFormatProvider, ICustomFormatter
{
   private const int ACCT_LENGTH = 12;

   public object GetFormat(Type formatType)
   {
      if (formatType == typeof(ICustomFormatter))
         return this;
      else
         return null;
   }

   public string Format(string fmt, object arg, IFormatProvider
formatProvider)
   {
      // Provide default formatting if arg is not an Int64.
      if (arg.GetType() != typeof(Int64))
         try {
            return HandleOtherFormats(fmt, arg);
         }
         catch (FormatException e) {
            throw new FormatException(String.Format("The format of '{0}' is
invalid.", fmt), e);
         }

      // Provide default formatting for unsupported format strings.
      string ufmt = fmt.ToUpper(CultureInfo.InvariantCulture);
      if (! (ufmt == "H" || ufmt == "I"))
         try {
            return HandleOtherFormats(fmt, arg);
         }
         catch (FormatException e) {
            throw new FormatException(String.Format("The format of '{0}' is
invalid.", fmt), e);
         }

      // Convert argument to a string.
```

```
      string result = arg.ToString();

      // If account number is less than 12 characters, pad with leading
zeroes.
      if (result.Length < ACCT_LENGTH)
         result = result.PadLeft(ACCT_LENGTH, '0');
      // If account number is more than 12 characters, truncate to 12
characters.
      if (result.Length > ACCT_LENGTH)
         result = result.Substring(0, ACCT_LENGTH);

      if (ufmt == "I")                    // Integer-only format.
         return result;
      // Add hyphens for H format specifier.
      else                                          // Hyphenated format.
         return result.Substring(0, 5) + "-" + result.Substring(5, 3) + "-"
+ result.Substring(8);
   }

   private string HandleOtherFormats(string format, object arg)
   {
      if (arg is IFormattable)
         return ((IFormattable)arg).ToString(format,
CultureInfo.CurrentCulture);
      else if (arg != null)
         return arg.ToString();
      else
         return String.Empty;
   }
}
```

The class that implements IFormatProvider can then be used in a call to a formatting and parsing operation. For example, the following code calls the String.Format(IFormatProvider, String, Object[]) method to generate a string that contains a formatted 12-digit account number.

C#

```
using System;
using System.Globalization;

public enum DaysOfWeek { Monday=1, Tuesday=2 };

public class TestFormatting
{
   public static void Main()
   {
      long acctNumber;
      double balance;
      DaysOfWeek wday;
      string output;
```

```
        acctNumber = 104254567890;
        balance = 16.34;
        wday = DaysOfWeek.Monday;

        output = String.Format(new AcctNumberFormat(),
                               "On {2}, the balance of account {0:H} was
{1:C2}.",
                               acctNumber, balance, wday);
        Console.WriteLine(output);

        wday = DaysOfWeek.Tuesday;
        output = String.Format(new AcctNumberFormat(),
                               "On {2}, the balance of account {0:I} was
{1:C2}.",
                               acctNumber, balance, wday);
        Console.WriteLine(output);
    }
}
// The example displays the following output:
//       On Monday, the balance of account 10425-456-7890 was $16.34.
//       On Tuesday, the balance of account 104254567890 was $16.34.
```

Remarks

The IFormatProvider interface supplies an object that provides formatting information for formatting and parsing operations. Formatting operations convert the value of a type to the string representation of that value. Typical formatting methods are the `ToString` methods of a type, as well as Format. Parsing operations convert the string representation of a value to a type with that value. Typical parsing methods are `Parse` and `TryParse`.

The IFormatProvider interface consists of a single method, IFormatProvider.GetFormat. GetFormat is a callback method: The parsing or formatting method calls it and passes it a Type object that represents the type of object that the formatting or parsing method expects will provide formatting information. The GetFormat method is responsible for returning an object of that type.

IFormatProvider implementations are often used implicitly by formatting and parsing methods. For example, the DateTime.ToString(String) method implicitly uses an IFormatProvider implementation that represents the system's current culture. IFormatProvider implementations can also be specified explicitly by methods that have a parameter of type IFormatProvider, such as Int32.Parse(String, IFormatProvider) and String.Format(IFormatProvider, String, Object[]).

The .NET Framework includes the following three predefined IFormatProvider implementations to provide culture-specific information that is used in formatting or

parsing numeric and date and time values:

- The NumberFormatInfo class, which provides information that is used to format numbers, such as the currency, thousands separator, and decimal separator symbols for a particular culture. For information about the predefined format strings recognized by a NumberFormatInfo object and used in numeric formatting operations, see Standard Numeric Format Strings and Custom Numeric Format Strings.

- The DateTimeFormatInfo class, which provides information that is used to format dates and times, such as the date and time separator symbols for a particular culture or the order and format of a date's year, month, and day components. For information about the predefined format strings recognized by a DateTimeFormatInfo object and used in numeric formatting operations, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.

- The CultureInfo class, which represents a particular culture. Its GetFormat method returns a culture-specific NumberFormatInfo or DateTimeFormatInfo object, depending on whether the CultureInfo object is used in a formatting or parsing operation that involves numbers or dates and times.

The .NET Framework also supports custom formatting. This typically involves the creation of a formatting class that implements both IFormatProvider and ICustomFormatter. An instance of this class is then passed as a parameter to a method that performs a custom formatting operation, such as String.Format(IFormatProvider, String, Object[]) The example provides an illustration of such a custom implementation that formats a number as a 12-digit account number.

Methods

Expand table

GetFormat(Type)	Returns an object that provides formatting services for the specified type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ICustomFormatter
- IFormattable
- Formatting Types in .NET
- How to: Define and Use Custom Numeric Format Providers

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IFormatProvider.GetFormat(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an object that provides formatting services for the specified type.

```
C#

public object? GetFormat (Type? formatType);
```

Parameters

`formatType` Type

An object that specifies the type of format object to return.

Returns

Object

An instance of the object specified by `formatType`, if the IFormatProvider implementation can supply that type of object; otherwise, `null`.

Examples

The following example illustrates the use of a class that implements the IFormatProvider interface and the GetFormat method. The `AcctNumberFormat` class converts an Int64 value that represents an account number to a formatted 12-digit account number. Its `GetFormat` method returns a reference to itself if the `formatType` parameter refers to a class that implements ICustomFormatter; otherwise, `GetFormat` returns `null`.

```
C#

public class AcctNumberFormat : IFormatProvider, ICustomFormatter
{
   private const int ACCT_LENGTH = 12;
```

```
   public object GetFormat(Type formatType)
   {
      if (formatType == typeof(ICustomFormatter))
         return this;
      else
         return null;
   }

   public string Format(string fmt, object arg, IFormatProvider
formatProvider)
   {
      // Provide default formatting if arg is not an Int64.
      if (arg.GetType() != typeof(Int64))
         try {
            return HandleOtherFormats(fmt, arg);
         }
         catch (FormatException e) {
            throw new FormatException(String.Format("The format of '{0}' is
invalid.", fmt), e);
         }

      // Provide default formatting for unsupported format strings.
      string ufmt = fmt.ToUpper(CultureInfo.InvariantCulture);
      if (! (ufmt == "H" || ufmt == "I"))
         try {
            return HandleOtherFormats(fmt, arg);
         }
         catch (FormatException e) {
            throw new FormatException(String.Format("The format of '{0}' is
invalid.", fmt), e);
         }

      // Convert argument to a string.
      string result = arg.ToString();

      // If account number is less than 12 characters, pad with leading
zeroes.
      if (result.Length < ACCT_LENGTH)
         result = result.PadLeft(ACCT_LENGTH, '0');
      // If account number is more than 12 characters, truncate to 12
characters.
      if (result.Length > ACCT_LENGTH)
         result = result.Substring(0, ACCT_LENGTH);

      if (ufmt == "I")                    // Integer-only format.
         return result;
      // Add hyphens for H format specifier.
      else                                          // Hyphenated format.
         return result.Substring(0, 5) + "-" + result.Substring(5, 3) + "-"
+ result.Substring(8);
   }

   private string HandleOtherFormats(string format, object arg)
   {
```

```
      if (arg is IFormattable)
         return ((IFormattable)arg).ToString(format,
CultureInfo.CurrentCulture);
      else if (arg != null)
         return arg.ToString();
      else
         return String.Empty;
   }
}
```

An instance of the `AcctNumberFormat` class can then be passed as an argument to a method that provides formatting or parsing services. For example, the following code passes an `AcctNumberFormat` class to the String.Format(IFormatProvider, String, Object[]) method to generate a formatted 12-digit account number.

C#

```
using System;
using System.Globalization;

public enum DaysOfWeek { Monday=1, Tuesday=2 };

public class TestFormatting
{
   public static void Main()
   {
      long acctNumber;
      double balance;
      DaysOfWeek wday;
      string output;

      acctNumber = 104254567890;
      balance = 16.34;
      wday = DaysOfWeek.Monday;

      output = String.Format(new AcctNumberFormat(),
                             "On {2}, the balance of account {0:H} was
{1:C2}.",
                             acctNumber, balance, wday);
      Console.WriteLine(output);

      wday = DaysOfWeek.Tuesday;
      output = String.Format(new AcctNumberFormat(),
                             "On {2}, the balance of account {0:I} was
{1:C2}.",
                             acctNumber, balance, wday);
      Console.WriteLine(output);
   }
}
// The example displays the following output:
```

```
//       On Monday, the balance of account 10425-456-7890 was $16.34.
//       On Tuesday, the balance of account 104254567890 was $16.34.
```

Remarks

GetFormat is a callback method that formatting and parsing methods invoke to retrieve information about the format of the input string in parsing operations or the format of the output string in formatting operations. In the `formatType` parameter, the formatting or parsing method passes the type of object it requires to perform its operation. If the IFormatProvider implementation can supply this formatting or parsing object, it returns that object. If not, it returns `null`.

For example, in the call to the Int32.ToString(IFormatProvider) method, the method argument is an IFormatProvider object that provides information about how the string representation of the current integer instance might be formatted. When the runtime executes the method, it calls the IFormatProvider object's GetFormat method and passes it a Type object that represents the NumberFormatInfo type. If the IFormatProvider object can supply a NumberFormatInfo object, it returns that object. If it cannot supply an object of that type, it returns `null`.

You can implement the IFormatProvider interface and the GetFormat method in classes that provide custom formatting or parsing services. The IFormatProvider implementation is then passed as an argument to any overload of a parsing or formatting method that has a parameter of type IFormatProvider, such as String.Format(IFormatProvider, String, Object[]), Int32.ToString(String, IFormatProvider), or DateTime.Parse(String, IFormatProvider).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IFormattable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides functionality to format the value of an object into a string representation.

C#

```
public interface IFormattable
```

Derived System.Byte
System.Char
System.DateOnly
System.DateTime
System.DateTimeOffset
More...

Examples

The following example defines a `Temperature` class that implements the IFormattable interface. The class supports four format specifiers: "G" and "C", which indicate that the temperature is to be displayed in Celsius; "F", which indicates that the temperature is to be displayed in Fahrenheit; and "K", which indicates that the temperature is to be displayed in Kelvin. In addition, the IFormattable.ToString implementation also can handle a format string that is `null` or empty. The other two `ToString` methods defined by the `Temperature` class simply wrap a call to the IFormattable.ToString implementation.

C#

```
using System;
using System.Globalization;

public class Temperature : IFormattable
{
   private decimal temp;

   public Temperature(decimal temperature)
```

```
    {
       if (temperature < -273.15m)
         throw new ArgumentOutOfRangeException(String.Format("{0} is less
than absolute zero.",
                                                 temperature));
       this.temp = temperature;
    }

    public decimal Celsius
    {
       get { return temp; }
    }

    public decimal Fahrenheit
    {
       get { return temp * 9 / 5 + 32; }
    }

    public decimal Kelvin
    {
       get { return temp + 273.15m; }
    }

    public override string ToString()
    {
       return this.ToString("G", CultureInfo.CurrentCulture);
    }

    public string ToString(string format)
    {
       return this.ToString(format, CultureInfo.CurrentCulture);
    }

    public string ToString(string format, IFormatProvider provider)
    {
       if (String.IsNullOrEmpty(format)) format = "G";
       if (provider == null) provider = CultureInfo.CurrentCulture;

       switch (format.ToUpperInvariant())
       {
          case "G":
          case "C":
             return temp.ToString("F2", provider) + " °C";
          case "F":
             return Fahrenheit.ToString("F2", provider) + " °F";
          case "K":
             return Kelvin.ToString("F2", provider) + " K";
          default:
             throw new FormatException(String.Format("The {0} format string
is not supported.", format));
       }
    }
}
```

The following example then calls the IFormattable.ToString implementation either directly or by using a composite format string.

C#

```
public class Example
{
   public static void Main()
   {
      // Use composite formatting with format string in the format item.
      Temperature temp1 = new Temperature(0);
      Console.WriteLine("{0:C} (Celsius) = {0:K} (Kelvin) = {0:F}
(Fahrenheit)\n", temp1);

      // Use composite formatting with a format provider.
      temp1 = new Temperature(-40);
      Console.WriteLine(String.Format(CultureInfo.CurrentCulture, "{0:C}
(Celsius) = {0:K} (Kelvin) = {0:F} (Fahrenheit)", temp1));
      Console.WriteLine(String.Format(new CultureInfo("fr-FR"), "{0:C}
(Celsius) = {0:K} (Kelvin) = {0:F} (Fahrenheit)\n", temp1));

      // Call ToString method with format string.
      temp1 = new Temperature(32);
      Console.WriteLine("{0} (Celsius) = {1} (Kelvin) = {2} (Fahrenheit)\n",
                        temp1.ToString("C"), temp1.ToString("K"),
temp1.ToString("F"));

      // Call ToString with format string and format provider
      temp1 = new Temperature(100)      ;
      NumberFormatInfo current = NumberFormatInfo.CurrentInfo;
      CultureInfo nl = new CultureInfo("nl-NL");
      Console.WriteLine("{0} (Celsius) = {1} (Kelvin) = {2} (Fahrenheit)",
                        temp1.ToString("C", current), temp1.ToString("K",
current), temp1.ToString("F", current));
      Console.WriteLine("{0} (Celsius) = {1} (Kelvin) = {2} (Fahrenheit)",
                        temp1.ToString("C", nl), temp1.ToString("K", nl),
temp1.ToString("F", nl));
   }
}
// The example displays the following output:
//    0.00 °C (Celsius) = 273.15 K (Kelvin) = 32.00 °F (Fahrenheit)
//
//    -40.00 °C (Celsius) = 233.15 K (Kelvin) = -40.00 °F (Fahrenheit)
//    -40,00 °C (Celsius) = 233,15 K (Kelvin) = -40,00 °F (Fahrenheit)
//
//    32.00 °C (Celsius) = 305.15 K (Kelvin) = 89.60 °F (Fahrenheit)
//
//    100.00 °C (Celsius) = 373.15 K (Kelvin) = 212.00 °F (Fahrenheit)
//    100,00 °C (Celsius) = 373,15 K (Kelvin) = 212,00 °F (Fahrenheit)
```

Remarks

The IFormattable interface converts an object to its string representation based on a format string and a format provider.

A format string typically defines the general appearance of an object. For example, the .NET Framework supports the following:

- Standard format strings for formatting enumeration values (see Enumeration Format Strings).
- Standard and custom format strings for formatting numeric values (see Standard Numeric Format Strings and Custom Numeric Format Strings).
- Standard and custom format strings for formatting date and time values (see Standard Date and Time Format Strings and Custom Date and Time Format Strings).
- Standard and custom format strings for formatting time intervals (see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings).

You can also define your own format strings to support formatting of your application-defined types.

A format provider returns a formatting object that typically defines the symbols used in converting an object to its string representation. For example, when you convert a number to a currency value, a format provider defines the currency symbol that appears in the result string. The .NET Framework defines three format providers:

- The System.Globalization.CultureInfo class, which returns either a NumberFormatInfo object for formatting numeric values, or a DateTimeFormatInfo object for formatting date and time values.
- The System.Globalization.NumberFormatInfo class, which returns an instance of itself for formatting numeric values.
- The System.Globalization.DateTimeFormatInfo class, which returns an instance of itself for formatting date and time values.

In addition, you can define your own custom format providers to supply culture-specific, profession-specific, or industry-specific information used in formatting. For more information about implementing custom formatting by using a custom format provider, see ICustomFormatter.

The IFormattable interface defines a single method, ToString, that supplies formatting services for the implementing type. The ToString method can be called directly. In addition, it is called automatically by the Convert.ToString(Object) and

Convert.ToString(Object, IFormatProvider) methods, and by methods that use the composite formatting feature in the .NET Framework. Such methods include Console.WriteLine(String, Object), String.Format, and StringBuilder.AppendFormat(String, Object), among others. The ToString method is called for each format item in the method's format string.

The IFormattable interface is implemented by the base data types.

Notes to Implementers

Classes that require more control over the formatting of strings than ToString() provides should implement IFormattable.

A class that implements IFormattable must support the "G" (general) format specifier. Besides the "G" specifier, the class can define the list of format specifiers that it supports. In addition, the class must be prepared to handle a format specifier that is `null`. For more information about formatting and formatting codes, see Formatting Types

Methods

Expand table

ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IFormatProvider

- ToString()
- CurrentCulture
- Formatting Types in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IFormattable.ToString(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Formats the value of the current instance using the specified format.

C#

```
public string ToString (string? format, IFormatProvider? formatProvider);
```

Parameters

`format` String

The format to use.

-or-

A null reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`formatProvider` IFormatProvider

The provider to use to format the value.

-or-

A null reference (`Nothing` in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

String

The value of the current instance in the specified format.

Examples

The following example demonstrates a `Temperature` class that implements the ToString method. This code example is part of a larger example provided for the IFormattable class.

C#

```
using System;
using System.Globalization;

public class Temperature : IFormattable
{
   private decimal temp;

   public Temperature(decimal temperature)
   {
      if (temperature < -273.15m)
        throw new ArgumentOutOfRangeException(String.Format("{0} is less
than absolute zero.",
                                                 temperature));
      this.temp = temperature;
   }

   public decimal Celsius
   {
      get { return temp; }
   }

   public decimal Fahrenheit
   {
      get { return temp * 9 / 5 + 32; }
   }

   public decimal Kelvin
   {
      get { return temp + 273.15m; }
   }

   public override string ToString()
   {
      return this.ToString("G", CultureInfo.CurrentCulture);
   }

   public string ToString(string format)
   {
      return this.ToString(format, CultureInfo.CurrentCulture);
   }

   public string ToString(string format, IFormatProvider provider)
   {
      if (String.IsNullOrEmpty(format)) format = "G";
      if (provider == null) provider = CultureInfo.CurrentCulture;

      switch (format.ToUpperInvariant())
```

```
        {
            case "G":
            case "C":
                return temp.ToString("F2", provider) + " °C";
            case "F":
                return Fahrenheit.ToString("F2", provider) + " °F";
            case "K":
                return Kelvin.ToString("F2", provider) + " K";
            default:
                throw new FormatException(String.Format("The {0} format string
is not supported.", format));
        }
    }
}
```

Remarks

The ToString method converts a value to a string representation that can be expressed in multiple ways. Its precise format depends on specific symbols or a specified order defined by specific cultures, professions, or industries. You can call the method directly. It is also called automatically by the Convert.ToString(Object) and Convert.ToString(Object, IFormatProvider) methods, and by methods that use the composite formatting feature in the .NET Framework, such as String.Format(String, Object[]), Console.WriteLine(String, Object[]), and StringBuilder.AppendFormat(String, Object[]). (For more information, see Composite Formatting.)

Composite formatting methods call the ToString method once for each format item in a format string. The parameters passed to the method depend on the specific formatting method that is called and on the content of the format item, as follows:

- If the format item does not include a format string (for example, if the format item is simply `{0}`), it is passed `null` as the value of the System.String parameter.
- If the format item includes a format string (for example, `{0:G}`), that format string is passed as the value of the System.String parameter.
- If the original method call does not include an System.IFormatProvider parameter, CultureInfo.CurrentCulture is passed as the value of the System.IFormatProvider parameter.
- If the original method call includes an System.IFormatProvider parameter, the provider that is supplied in the method call is passed as the value of the System.IFormatProvider parameter.

> **Note**
>
> An object's **ToString** implementation is called by composite formatting methods only if they are not passed an **ICustomFormatter** format provider, or if the **Format** method of the custom format provider returns `null`.

The .NET Framework includes three format providers, all of which implement the IFormatProvider interface:

- NumberFormatInfo supplies numeric formatting information, such as the characters to use for decimal and group separators, and the spelling and placement of currency symbols in monetary values.
- DateTimeFormatInfo supplies date-related and time-related formatting information, such as the position of the month, the day, and the year in a date pattern.
- CultureInfo contains the default formatting information in a specific culture, including the numeric format information, and date-related and time-related formatting information.

In addition, you can define your own custom format provider.

Notes to Implementers

The ToString(String, IFormatProvider) method must support the "G" (general) format specifier. Besides the "G" specifier, the class can define the list of format specifiers that it supports. In addition, the class must be prepared to handle a format specifier that is `null`. For more information about formatting and formatting codes, see Formatting Types.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String
- IFormatProvider
- NumberFormatInfo
- DateTimeFormatInfo
- CultureInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a type that can be used to index a collection either from the beginning or the end.

```
C#

public readonly struct Index : IEquatable<Index>
```

Inheritance Object → ValueType → Index

Implements IEquatable<Index>

Remarks

`Index` is used by the C# compiler to support the `^` or "index from end" operator:

```
C#

int[] someArray = new int[5] { 1, 2, 3, 4, 5 };
int lastElement = someArray[^1]; // lastElement = 5
```

Constructors

Expand table

Index(Int32, Boolean)	Initializes a new Index with a specified index position and a value that indicates if the index is from the beginning or the end of a collection.

Properties

Expand table

End	Gets an Index that points beyond the last element.
IsFromEnd	Gets a value that indicates whether the index is from the start or the end.
Start	Gets an Index that points to the first element of a collection.
Value	Gets the index value.

Methods

Expand table

Equals(Index)	Returns a value that indicates whether the current object is equal to another Index object.
Equals(Object)	Indicates whether the current Index object is equal to a specified object.
FromEnd(Int32)	Creates an Index from the end of a collection at a specified index position.
From Start(Int32)	Creates an Index from the specified index at the start of a collection.
GetHashCode()	Returns the hash code for this instance.
GetOffset(Int32)	Calculates the offset from the start of the collection using the specified collection length.
ToString()	Returns the string representation of the current Index instance.

Operators

Expand table

Implicit(Int32 to Index)	Converts an integer number to an Index.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Index(Int32, Boolean) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new Index with a specified index position and a value that indicates if the index is from the beginning or the end of a collection.

```
C#
public Index (int value, bool fromEnd = false);
```

Parameters

`value` Int32

The index value. It has to be greater then or equal to zero.

`fromEnd` Boolean

`true` to index from the end of the collection, or `false` to index from the beginning of the collection.

Remarks

If the Index is constructed from the end, an index value of 1 points to the last element, and an index value of 0 points beyond the last element.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Index.End Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an Index that points beyond the last element.

C#

```
public static Index End { get; }
```

Property Value

Index

An index that points beyond the last element.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.IsFromEnd Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the index is from the start or the end.

C#

```
public bool IsFromEnd { get; }
```

Property Value

Boolean

`true` if the Index is from the end; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.Start Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an Index that points to the first element of a collection.

C#

```
public static Index Start { get; }
```

Property Value

Index

An instance that points to the first element of a collection.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the index value.

```
C#
public int Value { get; }
```

Property Value

Int32

The index value.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Index)	Returns a value that indicates whether the current object is equal to another Index object.
Equals(Object)	Indicates whether the current Index object is equal to a specified object.

Equals(Index)

Returns a value that indicates whether the current object is equal to another Index object.

```
C#

public bool Equals (Index other);
```

Parameters

`other` Index

The object to compare with this instance.

Returns

Boolean

`true` if the current Index object is equal to `other`; `false` otherwise.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Equals(Object)

Indicates whether the current Index object is equal to a specified object.

C#

```
public override bool Equals (object? value);
```

Parameters

`value` Object

An object to compare with this instance.

Returns

Boolean

`true` if `value` is of type Index and is equal to the current instance; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Index.FromEnd(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an Index from the end of a collection at a specified index position.

C#

```
public static Index FromEnd (int value);
```

Parameters

`value` Int32

The index value from the end of a collection.

Returns

Index

The index value.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Index.FromStart(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an Index from the specified index at the start of a collection.

C#

```
public static Index FromStart (int value);
```

Parameters

`value` Int32

The index position from the start of a collection.

Returns

Index

The index value.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Index.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.GetOffset(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the offset from the start of the collection using the specified collection length.

```
C#
public int GetOffset (int length);
```

Parameters

`length` Int32

The length of the collection that the Index will be used with. Must be a positive value.

Returns

Int32

The offset.

Remarks

For performance reasons, this method does not validate if `length` or the returned value are negative. It also doesn't validate if the returned value is greater than `length`.

Collections are not expected to have a negative length/count. If this method's returned offset is negative and is then used to index a collection, the runtime will throw ArgumentOutOfRangeException, which will have the same effect as validation.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of the current Index instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the Index.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Index.Implicit(Int32 to Index) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts an integer number to an Index.

```
C#
public static implicit operator Index (int value);
```

Parameters

`value` Int32

The integer to convert.

Returns

Index

An index representing the integer.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IndexOutOfRangeException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt is made to access an element of an array or collection with an index that is outside its bounds.

C#

```
public sealed class IndexOutOfRangeException : SystemException
```

Inheritance Object → Exception → SystemException → IndexOutOfRangeException

Remarks

An IndexOutOfRangeException exception is thrown when an invalid index is used to access a member of an array or a collection, or to read or write from a particular location in a buffer. This exception inherits from the Exception class but adds no unique members.

Typically, an IndexOutOfRangeException exception is thrown as a result of developer error. Instead of handling the exception, you should diagnose the cause of the error and correct your code. The most common causes of the error are:

- Forgetting that the upper bound of a collection or a zero-based array is one less than its number of members or elements, as the following example illustrates.

  C#

  ```
  using System;
  using System.Collections.Generic;

  public class Example
  {
     public static void Main()
     {
        List<Char> characters = new List<Char>();
        characters.InsertRange(0, new Char[] { 'a', 'b', 'c', 'd', 'e',
  'f' } );
  ```

```
      for (int ctr = 0; ctr <= characters.Count; ctr++)
         Console.Write("'{0}'    ", characters[ctr]);
   }
}
// The example displays the following output:
//    'a'    'b'    'c'    'd'    'e'    'f'
//    Unhandled Exception:
//    System.ArgumentOutOfRangeException:
//    Index was out of range. Must be non-negative and less than the
size of the collection.
//    Parameter name: index
//       at Example.Main()
```

To correct the error, you can use code like the following.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      List<Char> characters = new List<Char>();
      characters.InsertRange(0, new Char[] { 'a', 'b', 'c', 'd', 'e',
'f' } );
      for (int ctr = 0; ctr < characters.Count; ctr++)
         Console.Write("'{0}'    ", characters[ctr]);
   }
}
// The example displays the following output:
//        'a'    'b'    'c'    'd'    'e'    'f'
```

Alternately, instead of iterating all the elements in the array by their index, you can use the `foreach` statement (in C#), the `for...in` statement (in F#), or the `For Each` statement (in Visual Basic).

- Attempting to assign an array element to another array that has not been adequately dimensioned and that has fewer elements than the original array. The following example attempts to assign the last element in the `value1` array to the same element in the `value2` array. However, the `value2` array has been incorrectly dimensioned to have six instead of seven elements. As a result, the assignment throws an IndexOutOfRangeException exception.

C#

```
public class Example
{
```

```
    public static void Main()
    {
       int[] values1 = { 3, 6, 9, 12, 15, 18, 21 };
       int[] values2 = new int[6];

       // Assign last element of the array to the new array.
       values2[values1.Length - 1] = values1[values1.Length - 1];
    }
}
// The example displays the following output:
//       Unhandled Exception:
//       System.IndexOutOfRangeException:
//       Index was outside the bounds of the array.
//       at Example.Main()
```

- Using a value returned by a search method to iterate a portion of an array or collection starting at a particular index position. If you forget to check whether the search operation found a match, the runtime throws an IndexOutOfRangeException exception, as shown in this example.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   static List<int> numbers = new List<int>();

   public static void Main()
   {
      int startValue;
      string[] args = Environment.GetCommandLineArgs();
      if (args.Length < 2)
         startValue = 2;
      else
         if (! Int32.TryParse(args[1], out startValue))
            startValue = 2;

      ShowValues(startValue);
   }

   private static void ShowValues(int startValue)
   {
      // Create a collection with numeric values.
      if (numbers.Count == 0)
         numbers.AddRange( new int[] { 2, 4, 6, 8, 10, 12, 14, 16, 18,
20, 22} );

      // Get the index of a startValue.
      Console.WriteLine("Displaying values greater than or equal to
{0}:",
```

```
                        startValue);
      int startIndex = numbers.IndexOf(startValue);
      // Display all numbers from startIndex on.
      for (int ctr = startIndex; ctr < numbers.Count; ctr++)
         Console.Write("    {0}", numbers[ctr]);
   }
}
// The example displays the following output if the user supplies
// 7 as a command-line parameter:
//    Displaying values greater than or equal to 7:
//
//    Unhandled Exception: System.ArgumentOutOfRangeException:
//    Index was out of range. Must be non-negative and less than the
size of the collection.
//    Parameter name: index
//       at System.Collections.Generic.List`1.get_Item(Int32 index)
//       at Example.ShowValues(Int32 startValue)
//       at Example.Main()
```

In this case, the List<T>.IndexOf method returns -1, which is an invalid index value, when it fails to find a match. To correct this error, check the search method's return value before iterating the array, as shown in this example.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   static List<int> numbers = new List<int>();

   public static void Main()
   {
      int startValue;
      string[] args = Environment.GetCommandLineArgs();
      if (args.Length < 2)
         startValue = 2;
      else
         if (! Int32.TryParse(args[1], out startValue))
            startValue = 2;

      ShowValues(startValue);
   }

   private static void ShowValues(int startValue)
   {
      // Create a collection with numeric values.
      if (numbers.Count == 0)
         numbers.AddRange( new int[] { 2, 4, 6, 8, 10, 12, 14, 16, 18,
20, 22} );
```

```
      // Get the index of startValue.
      int startIndex = numbers.IndexOf(startValue);
      if (startIndex < 0) {
         Console.WriteLine("Unable to find {0} in the collection.",
startValue);
      }
      else {
         // Display all numbers from startIndex on.
         Console.WriteLine("Displaying values greater than or equal to
{0}:",
                           startValue);
         for (int ctr = startIndex; ctr < numbers.Count; ctr++)
            Console.Write("    {0}", numbers[ctr]);
      }
   }
}
// The example displays the following output if the user supplies
// 7 as a command-line parameter:
//      Unable to find 7 in the collection.
```

- Trying to use or enumerate a result set, collection, or array returned by a query without testing whether the returned object has any valid data.

- Using a computed value to define the starting index, the ending index, or the number of items to be iterated. If the result of the computation is unexpected, it might result in an IndexOutOfRangeException exception. You should check your program's logic in calculating the index value and validate the value before iterating the array or collection. The following conditions must all be true; otherwise, an IndexOutOfRangeException exception is thrown:

  - The starting index must be greater than or equal to Array.GetLowerBound for the dimension of the array that you want to iterate, or greater than or equal to 0 for a collection.

  - The ending index cannot exceed Array.GetUpperBound for the dimension of the array that you want to iterate, or cannot be greater than or equal to the `Count` property of a collection.

  - The following equation must be true for the dimension of the array that you want to iterate:

    ```
    start_index >= lower_bound And start_index + items_to_iterate - 1 <=
    upper_bound
    ```

    For a collection, the following equation must be true:

```
start_index >= 0 And start_index + items_to_iterate <= Count
```

> **Tip**
>
> The starting index of an array or collection can never be a negative number.

- Assuming that an array must be zero-based. Arrays that are not zero-based can be created by the Array.CreateInstance(Type, Int32[], Int32[]) method and can be returned by COM interop, although they aren't CLS-compliant. The following example illustrates the IndexOutOfRangeException that is thrown when you try to iterate a non-zero-based array created by the Array.CreateInstance(Type, Int32[], Int32[]) method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Array values = Array.CreateInstance(typeof(int), new int[] { 10
},
                                          new int[] { 1 });
      int value = 2;
      // Assign values.
      for (int ctr = 0; ctr < values.Length; ctr++) {
         values.SetValue(value, ctr);
         value *= 2;
      }

      // Display values.
      for (int ctr = 0; ctr < values.Length; ctr++)
         Console.Write("{0}    ", values.GetValue(ctr));
   }
}
// The example displays the following output:
//    Unhandled Exception:
//    System.IndexOutOfRangeException: Index was outside the bounds of
the array.
//       at System.Array.InternalGetReference(Void* elemRef, Int32
rank, Int32* pIndices)
//       at System.Array.SetValue(Object value, Int32 index)
//       at Example.Main()
```

To correct the error, as the following example does, you can call the GetLowerBound method instead of making assumptions about the starting index of an array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Array values = Array.CreateInstance(typeof(int), new int[] { 10
},
                                          new int[] { 1 });
      int value = 2;
      // Assign values.
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++) {
         values.SetValue(value, ctr);
         value *= 2;
      }

      // Display values.
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++)
         Console.Write("{0}    ", values.GetValue(ctr));
   }
}
// The example displays the following output:
//        2    4    8    16    32    64    128    256    512    1024
```

Note that when you call the GetLowerBound method to get the starting index of an array, you should also call the Array.GetUpperBound(Int32) method to get its ending index.

- Confusing an index and the value at that index in a numeric array or collection. This issue usually occurs when using the `foreach` statement (in C#), the `for...in` statement (in F#), or the `For Each` statement (in Visual Basic). The following example illustrates the problem.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Generate array of random values.
```

```
        int[] values = PopulateArray(5, 10);
        // Display each element in the array.
        foreach (var value in values)
           Console.Write("{0}   ", values[value]);
    }

    private static int[] PopulateArray(int items, int maxValue)
    {
        int[] values = new int[items];
        Random rnd = new Random();
        for (int ctr = 0; ctr < items; ctr++)
           values[ctr] = rnd.Next(0, maxValue + 1);

        return values;
    }
}
// The example displays output like the following:
//    6   4   4
//    Unhandled Exception: System.IndexOutOfRangeException:
//    Index was outside the bounds of the array.
//       at Example.Main()
```

The iteration construct returns each value in an array or collection, not its index. To eliminate the exception, use this code.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        // Generate array of random values.
        int[] values = PopulateArray(5, 10);
        // Display each element in the array.
        foreach (var value in values)
           Console.Write("{0}   ", value);
    }

    private static int[] PopulateArray(int items, int maxValue)
    {
        int[] values = new int[items];
        Random rnd = new Random();
        for (int ctr = 0; ctr < items; ctr++)
           values[ctr] = rnd.Next(0, maxValue + 1);

        return values;
    }
}
// The example displays output like the following:
//        10   6   7   5   8
```

- Providing an invalid column name to the DataView.Sort property.
- Violating thread safety. Operations such as reading from the same StreamReader object, writing to the same StreamWriter object from multiple threads, or enumerating the objects in a Hashtable from different threads can throw an IndexOutOfRangeException if the object isn't accessed in a thread-safe way. This exception is typically intermittent because it relies on a race condition.

Using hard-coded index values to manipulate an array is likely to throw an exception if the index value is incorrect or invalid, or if the size of the array being manipulation is unexpected. To prevent an operation from throwing an IndexOutOfRangeException exception, you can do the following:

- Iterate the elements of the array using the foreach statement (in C#), the for...in statement (in F#), or the For Each...Next construct (in Visual Basic) instead of iterating elements by index.
- Iterate the elements by index starting with the index returned by the Array.GetLowerBound method and ending with the index returned by the Array.GetUpperBound method.
- If you are assigning elements in one array to another, ensure that the target array has at least as many elements as the source array by comparing their Array.Length properties.

For a list of initial property values for an instance of IndexOutOfRangeException, see the IndexOutOfRangeException constructors.

The following intermediate language (IL) instructions throw IndexOutOfRangeException:

- ldelem.<type>
- ldelema
- stelem.<type>

IndexOutOfRangeException uses the HRESULT COR_E_INDEXOUTOFRANGE, which has the value 0x80131508.

Constructors

Expand table

IndexOutOfRangeException()	Initializes a new instance of the IndexOutOfRangeException class.
IndexOutOfRangeException(String)	Initializes a new instance of the IndexOutOfRangeException class with a specified error message.
IndexOutOfRangeException(String, Exception)	Initializes a new instance of the IndexOutOfRangeException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.

	(Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IndexOutOfRangeException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the IndexOutOfRangeException class.

Overloads

Expand table

IndexOutOfRangeException()	Initializes a new instance of the IndexOutOfRangeException class.
IndexOutOfRangeException(String)	Initializes a new instance of the IndexOutOfRangeException class with a specified error message.
IndexOutOfRangeException(String, Exception)	Initializes a new instance of the IndexOutOfRangeException class with a specified error message and a reference to the inner exception that is the cause of this exception.

IndexOutOfRangeException()

Initializes a new instance of the IndexOutOfRangeException class.

C#

```
public IndexOutOfRangeException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "An array index is out of range."

This message takes into account the current system culture.

The following table shows the initial property values for an instance of IndexOutOfRangeException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOutOfRangeException(String)

Initializes a new instance of the IndexOutOfRangeException class with a specified error message.

C#

```
public IndexOutOfRangeException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of IndexOutOfRangeException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOutOfRangeException(String, Exception)

Initializes a new instance of the IndexOutOfRangeException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public IndexOutOfRangeException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The `InnerException` property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the `InnerException` property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of IndexOutOfRangeException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InsufficientExecutionStackException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is insufficient execution stack available to allow most methods to execute.

C#

```
public sealed class InsufficientExecutionStackException : SystemException
```

Inheritance Object → Exception → SystemException → InsufficientExecutionStackException

Constructors

Expand table

InsufficientExecutionStackException()	Initializes a new instance of the InsufficientExecutionStackException class.
InsufficientExecutionStackException(String)	Initializes a new instance of the InsufficientExecutionStackException class with a specified error message.
InsufficientExecutionStackException(String, Exception)	Initializes a new instance of the InsufficientExecutionStackException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)

MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InsufficientExecutionStackException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InsufficientExecutionStackException class.

Overloads

Expand table

InsufficientExecutionStackException()	Initializes a new instance of the InsufficientExecutionStackException class.
InsufficientExecutionStackException(String)	Initializes a new instance of the InsufficientExecutionStackException class with a specified error message.
InsufficientExecutionStackException(String, Exception)	Initializes a new instance of the InsufficientExecutionStackException class with a specified error message and a reference to the inner exception that is the cause of this exception.

InsufficientExecutionStackException()

Initializes a new instance of the InsufficientExecutionStackException class.

C#

```
public InsufficientExecutionStackException ();
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InsufficientExecutionStackException(String)

Initializes a new instance of the InsufficientExecutionStackException class with a specified error message.

C#

```
public InsufficientExecutionStackException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InsufficientExecutionStackException(String, Exception)

Initializes a new instance of the InsufficientExecutionStackException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public InsufficientExecutionStackException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

InsufficientMemoryException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a check for sufficient available memory fails. This class cannot be inherited.

C#

```
public sealed class InsufficientMemoryException : OutOfMemoryException
```

Inheritance Object → Exception → SystemException → OutOfMemoryException → InsufficientMemoryException

Remarks

Unlike OutOfMemoryException, InsufficientMemoryException is thrown before starting an operation, and thus does not imply state corruption. An application can catch this exception, throttle back its memory usage, and avoid actual out of memory conditions and their potential for corrupting program state.

This exception is thrown by the MemoryFailPoint constructor when you specify a projected memory allocation larger than the amount of currently available memory. Your program can catch the exception and either delay the task that needs the memory or execute it in smaller steps that require less memory.

See the MemoryFailPoint class for a code example.

> **Note**
>
> This exception is not included in the **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **OutOfMemoryException** instead.

Constructors

Expand table

InsufficientMemoryException()	Initializes a new instance of the InsufficientMemoryException class with a system-supplied message that describes the error.
InsufficientMemoryException(String)	Initializes a new instance of the InsufficientMemoryException class with a specified message that describes the error.
InsufficientMemoryException(String, Exception)	Initializes a new instance of the InsufficientMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
InnerException	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

See also

- MemoryFailPoint

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InsufficientMemoryException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InsufficientMemoryException class.

Overloads

Expand table

InsufficientMemoryException()	Initializes a new instance of the InsufficientMemoryException class with a system-supplied message that describes the error.
InsufficientMemoryException(String)	Initializes a new instance of the InsufficientMemoryException class with a specified message that describes the error.
InsufficientMemoryException(String, Exception)	Initializes a new instance of the InsufficientMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

InsufficientMemoryException()

Initializes a new instance of the InsufficientMemoryException class with a system-supplied message that describes the error.

C#

```
public InsufficientMemoryException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Insufficient available memory to meet the expected demands of an operation at this time. Please try again later." This message takes into account the current system culture.

The following table shows the initial property values for an instance of InsufficientMemoryException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InsufficientMemoryException(String)

Initializes a new instance of the InsufficientMemoryException class with a specified message that describes the error.

C#

```
public InsufficientMemoryException (string? message);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of InsufficientMemoryException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InsufficientMemoryException(String, Exception)

Initializes a new instance of the InsufficientMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public InsufficientMemoryException (string? message, Exception?
```

```
innerException);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of InsufficientMemoryException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 128-bit signed integer.

C#

```
public readonly struct Int128 : IComparable<Int128>, IEquatable<Int128>,
IParsable<Int128>, ISpanParsable<Int128>, IUtf8SpanParsable<Int128>,
System.Numerics.IAdditionOperators<Int128,Int128,Int128>,
System.Numerics.IAdditiveIdentity<Int128,Int128>,
System.Numerics.IBinaryInteger<Int128>,
System.Numerics.IBinaryNumber<Int128>,
System.Numerics.IBitwiseOperators<Int128,Int128,Int128>,
System.Numerics.IComparisonOperators<Int128,Int128,bool>,
System.Numerics.IDecrementOperators<Int128>,
System.Numerics.IDivisionOperators<Int128,Int128,Int128>,
System.Numerics.IEqualityOperators<Int128,Int128,bool>,
System.Numerics.IIncrementOperators<Int128>,
System.Numerics.IMinMaxValue<Int128>,
System.Numerics.IModulusOperators<Int128,Int128,Int128>,
System.Numerics.IMultiplicativeIdentity<Int128,Int128>,
System.Numerics.IMultiplyOperators<Int128,Int128,Int128>,
System.Numerics.INumber<Int128>, System.Numerics.INumberBase<Int128>,
System.Numerics.IShiftOperators<Int128,int,Int128>,
System.Numerics.ISignedNumber<Int128>,
System.Numerics.ISubtractionOperators<Int128,Int128,Int128>,
System.Numerics.IUnaryNegationOperators<Int128,Int128>,
System.Numerics.IUnaryPlusOperators<Int128,Int128>
```

Inheritance Object → ValueType → Int128

Implements IComparable , IComparable<Int128> , IComparable<TSelf> , IEquatable<Int128> , IEquatable<TSelf> , IFormattable , IParsable<Int128> , IParsable<TSelf> , ISpanFormattable , ISpanParsable<Int128> , ISpanParsable<TSelf> , IAdditionOperators<Int128,Int128,Int128> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Int128,Int128> ,

IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Int128> ,

IBinaryNumber<Int128> , IBinaryNumber<TSelf> ,

IBitwiseOperators<Int128,Int128,Int128> ,

IBitwiseOperators<TSelf,TSelf,TSelf> ,

IComparisonOperators<Int128,Int128,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<Int128> , IDecrementOperators<TSelf> ,

IDivisionOperators<Int128,Int128,Int128> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<Int128,Int128,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Int128> ,

IIncrementOperators<TSelf> , IMinMaxValue<Int128> ,

IModulusOperators<Int128,Int128,Int128> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<Int128,Int128> , IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<Int128,Int128,Int128> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Int128> ,

INumber<TSelf> , INumberBase<Int128> , INumberBase<TSelf> ,

IShiftOperators<Int128,Int32,Int128> , IShiftOperators<TSelf,Int32,TSelf> ,

ISignedNumber<Int128> , ISubtractionOperators<Int128,Int128,Int128> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<Int128,Int128> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Int128,Int128> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Int128> ,

IUtf8SpanParsable<TSelf>

Constructors

Expand table

Int128(UInt64, UInt64)	Initializes a new instance of the Int128 struct.

Properties

Expand table

MaxValue	Gets the maximum value of the current type.
MinValue	Gets the minimum value of the current type.
NegativeOne	Gets the value `-1` for the type.
One	Gets the value `1` for the type.
Zero	Gets the value `0` for the type.

Methods

Expand table

Abs(Int128)	Computes the absolute of a value.
Clamp(Int128, Int128, Int128)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Int128)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CopySign(Int128, Int128)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(Int128, Int128)	Computes the quotient and remainder of two values.

Equals(Int128)	Indicates whether the current object is equal to another object of the same type.
Equals(Object)	Determines whether the specified object is equal to the current object.
GetHashCode()	Serves as the default hash function.
IsEvenInteger(Int128)	Determines if a value represents an even integral number.
IsNegative(Int128)	Determines if a value is negative.
IsOddInteger(Int128)	Determines if a value represents an odd integral number.
IsPositive(Int128)	Determines if a value is positive.
IsPow2(Int128)	Determines if a value is a power of two.
LeadingZeroCount(Int128)	Computes the number of leading zeros in a value.
Log2(Int128)	Computes the log2 of a value.
Max(Int128, Int128)	Compares two values to compute which is greater.
MaxMagnitude(Int128, Int128)	Compares two values to compute which is greater.
Min(Int128, Int128)	Compares two values to compute which is lesser.
MinMagnitude(Int128, Int128)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
Parse(String)	Parses a string into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(String, NumberStyles)	Parses a string into a value.
Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.
PopCount(Int128)	Computes the number of bits that are set in a value.

RotateLeft(Int128, Int32)	Rotates a value left by a given amount.
RotateRight(Int128, Int32)	Rotates a value right by a given amount.
Sign(Int128)	Computes the sign of a value.
ToString()	Returns a string that represents the current object.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
TrailingZeroCount(Int128)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int128)	Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit signed integer equivalent.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, Int128)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int128)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Int128)	Tries to parse a string into a value.

TryParse(String, Int128)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Int128)	Tries to parse a string into a value.

Operators

Expand table

Addition(Int128, Int128)	Adds two values together to compute their sum.
BitwiseAnd(Int128, Int128)	Computes the bitwise-and of two values.
BitwiseOr(Int128, Int128)	Computes the bitwise-or of two values.
CheckedAddition(Int128, Int128)	Adds two values together to compute their sum.
CheckedDecrement(Int128)	Decrements a value.
CheckedDivision(Int128, Int128)	Divides two values together to compute their quotient.
CheckedExplicit(Double)	
CheckedExplicit(Int128)	
CheckedExplicit(Single)	
CheckedIncrement(Int128)	Increments a value.
CheckedMultiply(Int128, Int128)	Multiplies two values together to compute their product.
CheckedSubtraction(Int128, Int128)	Subtracts two values to compute their difference.
CheckedUnaryNegation(Int128)	Computes the unary negation of a value.
Decrement(Int128)	Decrements a value.
Division(Int128, Int128)	Divides two values together to compute their quotient.
Equality(Int128, Int128)	Compares two values to determine equality.
ExclusiveOr(Int128, Int128)	Computes the exclusive-or of two values.
Explicit(Decimal to Int128)	Explicitly converts a Decimal value to a 128-bit signed integer.
Explicit(Double to Int128)	Explicitly converts a Double value to a 128-bit signed integer.
Explicit(Int128 to Byte)	Explicitly converts a 128-bit signed integer to a Byte value.

Explicit(Int128 to Char)	Explicitly converts a 128-bit signed integer to a Char value.
Explicit(Int128 to Decimal)	Explicitly converts a 128-bit signed integer to a Decimal value.
Explicit(Int128 to Double)	Explicitly converts a 128-bit signed integer to a Double value.
Explicit(Int128 to Half)	Explicitly converts a 128-bit signed integer to a Half value.
Explicit(Int128 to Int16)	Explicitly converts a 128-bit signed integer to a Int16 value.
Explicit(Int128 to Int32)	Explicitly converts a 128-bit signed integer to a Int32 value.
Explicit(Int128 to Int64)	Explicitly converts a 128-bit signed integer to a Int64 value.
Explicit(Int128 to IntPtr)	Explicitly converts a 128-bit signed integer to a IntPtr value.
Explicit(Int128 to SByte)	Explicitly converts a 128-bit signed integer to a SByte value.
Explicit(Int128 to Single)	Explicitly converts a 128-bit signed integer to a Single value.
Explicit(Int128 to UInt128)	Explicitly converts a 128-bit signed integer to a UInt128 value.
Explicit(Int128 to UInt16)	Explicitly converts a 128-bit signed integer to a UInt16 value.
Explicit(Int128 to UInt32)	Explicitly converts a 128-bit signed integer to a UInt32 value.
Explicit(Int128 to UInt64)	Explicitly converts a 128-bit signed integer to a UInt64 value.
Explicit(Int128 to UIntPtr)	Explicitly converts a 128-bit signed integer to a UIntPtr value.
Explicit(Single to Int128)	Explicitly converts a Single value to a 128-bit signed integer.
GreaterThan(Int128, Int128)	Compares two values to determine which is greater.
GreaterThanOrEqual(Int128, Int128)	Compares two values to determine which is greater or equal.
Implicit(Byte to Int128)	Implicitly converts a Byte value to a 128-bit signed integer.
Implicit(Char to Int128)	Implicitly converts a Char value to a 128-bit signed integer.
Implicit(Int16 to Int128)	Implicitly converts a Int16 value to a 128-bit signed integer.
Implicit(Int32 to Int128)	Implicitly converts a Int32 value to a 128-bit signed integer.
Implicit(Int64 to Int128)	Implicitly converts a Int64 value to a 128-bit signed integer.
Implicit(IntPtr to Int128)	Implicitly converts a IntPtr value to a 128-bit signed integer.
Implicit(SByte to Int128)	Implicitly converts a SByte value to a 128-bit signed integer.

Implicit(UInt16 to Int128)	Implicitly converts a UInt16 value to a 128-bit signed integer.
Implicit(UInt32 to Int128)	Implicitly converts a UInt32 value to a 128-bit signed integer.
Implicit(UInt64 to Int128)	Implicitly converts a UInt64 value to a 128-bit signed integer.
Implicit(UIntPtr to Int128)	Implicitly converts a UIntPtr value to a 128-bit signed integer.
Increment(Int128)	Increments a value.
Inequality(Int128, Int128)	Compares two values to determine inequality.
LeftShift(Int128, Int32)	Shifts a value left by a given amount.
LessThan(Int128, Int128)	Compares two values to determine which is less.
LessThanOrEqual(Int128, Int128)	Compares two values to determine which is less or equal.
Modulus(Int128, Int128)	Divides two values together to compute their modulus or remainder.
Multiply(Int128, Int128)	Multiplies two values together to compute their product.
OnesComplement(Int128)	Computes the ones-complement representation of a given value.
RightShift(Int128, Int32)	Shifts a value right by a given amount.
Subtraction(Int128, Int128)	Subtracts two values to compute their difference.
UnaryNegation(Int128)	Computes the unary negation of a value.
UnaryPlus(Int128)	Computes the unary plus of a value.
UnsignedRightShift(Int128, Int32)	Shifts a value right by a given amount.

Explicit Interface Implementations

Expand table

IAdditiveIdentity<Int128,Int128>.Additive Identity	Gets the additive identity of the current type.
IBinaryInteger<Int128>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Int128>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Int128>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Int128)	

IBinaryInteger<Int128>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Int128)	
IBinaryInteger<Int128>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Int128>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Int128>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IMultiplicativeIdentity<Int128,Int128>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
INumber<Int128>.MaxNumber(Int128, Int128)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<Int128>.MinNumber(Int128, Int128)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Int128>.IsCanonical(Int128)	Determines if a value is in its canonical representation.
INumberBase<Int128>.IsComplexNumber(Int128)	Determines if a value represents a complex number.
INumberBase<Int128>.IsFinite(Int128)	Determines if a value is finite.
INumberBase<Int128>.IsImaginaryNumber(Int128)	Determines if a value represents a pure imaginary number.
INumberBase<Int128>.IsInfinity(Int128)	Determines if a value is infinite.
INumberBase<Int128>.IsInteger(Int128)	Determines if a value represents an integral number.
INumberBase<Int128>.IsNaN(Int128)	Determines if a value is NaN.
INumberBase<Int128>.IsNegativeInfinity(Int128)	Determines if a value is negative infinity.
INumberBase<Int128>.IsNormal(Int128)	Determines if a value is normal.
INumberBase<Int128>.IsPositiveInfinity(Int128)	Determines if a value is positive infinity.
INumberBase<Int128>.IsRealNumber(Int128)	Determines if a value represents a real number.
INumberBase<Int128>.IsSubnormal(Int128)	Determines if a value is subnormal.
INumberBase<Int128>.IsZero(Int128)	Determines if a value is zero.

INumberBase<Int128>.MaxMagnitudeNumber(Int128, Int128)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Int128>.MinMagnitudeNumber(Int128, Int128)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Int128>.Radix	Gets the radix, or base, for the type.
INumberBase<Int128>.TryConvertFromChecked<TOther>(TOther, Int128)	
INumberBase<Int128>.TryConvertFromSaturating<TOther>(TOther, Int128)	
INumberBase<Int128>.TryConvertFromTruncating<TOther>(TOther, Int128)	
INumberBase<Int128>.TryConvertToChecked<TOther>(Int128, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Int128>.TryConvertToSaturating<TOther>(Int128, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Int128>.TryConvertToTruncating<TOther>(Int128, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int128(UInt64, UInt64) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the Int128 struct.

```
C#

[System.CLSCompliant(false)]
public Int128 (ulong upper, ulong lower);
```

Parameters

upper UInt64

The upper 64-bits of the 128-bit value.

lower UInt64

The lower 64-bits of the 128-bit value.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
public static Int128 MaxValue { get; }
```

Property Value

Int128

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
public static Int128 MinValue { get; }
```

Property Value

Int128

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

```
C#
public static Int128 NegativeOne { get; }
```

Property Value

Int128

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
public static Int128 One { get; }
```

Property Value

Int128

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
public static Int128 Zero { get; }
```

Property Value

Int128

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.Abs(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static Int128 Abs (Int128 value);
```

Parameters

`value` Int128

The value for which to get its absolute.

Returns

Int128

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Clamp(Int128, Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static Int128 Clamp (Int128 value, Int128 min, Int128 max);
```

Parameters

`value` Int128

The value to clamp.

`min` Int128

The inclusive minimum to which `value` should clamp.

`max` Int128

The inclusive maximum to which `value` should clamp.

Returns

Int128

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(Int128)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

CompareTo(Int128)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (Int128 value);
```

Parameters

`value` Int128

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CompareTo(Object)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.CopySign(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
public static Int128 CopySign (Int128 value, Int128 sign);
```

Parameters

`value` Int128

The value whose magnitude is used in the result.

`sign` Int128

The value whose sign is used in the result.

Returns

Int128

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static Int128 CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int128

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static Int128 CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

Int128

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static Int128 CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

Int128

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.DivRem(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (Int128 Quotient, Int128 Remainder) DivRem (Int128 left,
Int128 right);
```

Parameters

`left` Int128

The value which `right` divides.

`right` Int128

The value which divides `left`.

Returns

ValueTuple<Int128,Int128>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Int128)	Indicates whether the current object is equal to another object of the same type.
Equals(Object)	Determines whether the specified object is equal to the current object.

Equals(Int128)

Indicates whether the current object is equal to another object of the same type.

```
C#

public bool Equals (Int128 other);
```

Parameters

other Int128

An object to compare with this object.

Returns

Boolean

`true` if the current object is equal to the `other` parameter; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Equals(Object)

Determines whether the specified object is equal to the current object.

```
C#
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with the current object.

Returns

Boolean

`true` if the specified object is equal to the current object; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int128.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as the default hash function.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A hash code for the current object.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.IsEvenInteger(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (Int128 value);
```

Parameters

value Int128

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IsNegative(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IsOddInteger(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (Int128 value);
```

Parameters

value Int128

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IsPositive(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IsPow2(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.LeadingZeroCount(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

```
C#
public static Int128 LeadingZeroCount (Int128 value);
```

Parameters

`value` Int128

The value whose leading zeroes are to be counted.

Returns

Int128

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Log2(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static Int128 Log2 (Int128 value);
```

Parameters

`value` Int128

The value whose log2 is to be computed.

Returns

Int128

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Max(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static Int128 Max (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.MaxMagnitude(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static Int128 MaxMagnitude (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Min(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static Int128 Min (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.MinMagnitude(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static Int128 MinMagnitude (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Parses a string into a value.
Parse(String, NumberStyles)	Parses a string into a value.

Parse(String, NumberStyles, IFormatProvider)

Parses a string into a value.

C#

```
public static Int128 Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int128

The result of parsing `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static Int128 Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int128

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static Int128 Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int128

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Parses a string into a value.

C#

```
public static Int128 Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int128

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static Int128 Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int128

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static Int128 Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int128

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

Parses a string into a value.

C#

```
public static Int128 Parse (string s);
```

Parameters

s String

A string containing a number to parse.

Returns

Int128

The result of parsing `s`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles)

Parses a string into a value.

C#

```
public static Int128 Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

Returns

Int128

The result of parsing `s`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.PopCount(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static Int128 PopCount (Int128 value);
```

Parameters

`value` Int128

The value whose set bits are to be counted.

Returns

Int128

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.RotateLeft(Int128, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static Int128 RotateLeft (Int128 value, int rotateAmount);
```

Parameters

`value` Int128

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Int128

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.RotateRight(Int128, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
public static Int128 RotateRight (Int128 value, int rotateAmount);
```

Parameters

`value` Int128

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Int128

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Sign(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (Int128 value);
```

Parameters

`value` Int128

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
ToString()	Returns a string that represents the current object.

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

The format to use, or a `null` reference to use the default format defined for the type of the IFormattable implementation.

Returns

String

The string representation of the value of this instance as specified by `format`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString(String, IFormatProvider)

Formats the value of the current instance using the specified format.

```
C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

The format to use, or a `null` reference to use the default format defined for the type of the IFormattable implementation.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString()

Returns a string that represents the current object.

```
C#
```

```
public override string ToString ();
```

Returns

String

A string that represents the current object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.TrailingZeroCount(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

```
C#
public static Int128 TrailingZeroCount (Int128 value);
```

Parameters

`value` Int128

The value whose trailing zeroes are to be counted.

Returns

Int128

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Int128.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(String, Int128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, Int128)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Byte>, Int128)	Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit signed integer equivalent.
TryParse(String, NumberStyles, IFormatProvider, Int128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int128)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Int128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int128)	Tries to parse a string into a value.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int128)

Tries to parse a span of UTF-8 characters into a value.

```
C#
```

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out Int128 result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int128

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, Int128)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, out Int128 result);
```

Parameters

`s` String

The string to parse.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, Int128)

Tries to parse a span of characters into a value.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out Int128 result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, Int128)

Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit signed integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out Int128
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Int128

When this method returns, contains the 128-bit signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, NumberStyles, IFormatProvider, Int128)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out Int128 result);
```

Parameters

`s` String

The string to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int128)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
Int128 result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int128

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int128)

Tries to parse a span of characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
Int128 result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Int128)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
Int128 result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, Int128)

Tries to parse a string into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out Int128 result);
```

Parameters

`s` ReadOnlySpan<Char>

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Addition(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#
public static Int128 operator + (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to which `right` is added.

`right` Int128

The value which is added to `left`.

Returns

Int128

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.BitwiseAnd(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#
public static Int128 operator & (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to `and` with `right`.

`right` Int128

The value to `and` with `left`.

Returns

Int128

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.BitwiseOr(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#
public static Int128 operator | (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to `or` with `right`.

`right` Int128

The value to `or` with `left`.

Returns

Int128

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedAddition(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#
public static Int128 op_CheckedAddition (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to which `right` is added.

`right` Int128

The value which is added to `left`.

Returns

Int128

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedDecrement(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
public static Int128 op_CheckedDecrement (Int128 value);
```

Parameters

`value` Int128

The value to decrement.

Returns

Int128

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedDivision(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their quotient.

```
C#
public static Int128 op_CheckedDivision (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value which `right` divides.

`right` Int128

The value which divides `left`.

Returns

Int128

The quotient of `left` divided-by `right`.

Implements

CheckedDivision(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedExplicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CheckedExplicit(Single)
CheckedExplicit(Int128)
CheckedExplicit(Double)

CheckedExplicit(Single)

C#

```
public static Int128 op_CheckedExplicit (float value);
```

Parameters

`value` Single

Returns

Int128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Int128)

C#

```
public static byte op_CheckedExplicit (Int128 value);
```

Parameters

`value` Int128

Returns

Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Double)

C#

```
public static Int128 op_CheckedExplicit (double value);
```

Parameters

`value` Double

Returns

Int128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedIncrement(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
public static Int128 op_CheckedIncrement (Int128 value);
```

Parameters

`value` Int128

The value to increment.

Returns

Int128

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedMultiply(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#
public static Int128 op_CheckedMultiply (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value which `right` multiplies.

`right` Int128

The value which multiplies `left`.

Returns

Int128

The product of `left` multiplied-by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedSubtraction(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#
public static Int128 op_CheckedSubtraction (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value from which `right` is subtracted.

`right` Int128

The value which is subtracted from `left`.

Returns

Int128

The difference of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.CheckedUnaryNegation(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
public static Int128 op_CheckedUnaryNegation (Int128 value);
```

Parameters

`value` Int128

The value for which to compute the unary negation.

Returns

Int128

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Decrement(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
public static Int128 operator -- (Int128 value);
```

Parameters

`value` Int128

The value to decrement.

Returns

Int128

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Division(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their quotient.

```
C#
public static Int128 operator / (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value which `right` divides.

`right` Int128

The value which divides `left`.

Returns

Int128

The quotient of `left` divided-by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Equality(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#
public static bool operator == (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.ExclusiveOr(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#
public static Int128 operator ^ (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to xor with `right`.

`right` Int128

The value to xor with `left`.

Returns

Int128

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Explicit(Single to Int128)	Explicitly converts a Single value to a 128-bit signed integer.
Explicit(Int128 to UIntPtr)	Explicitly converts a 128-bit signed integer to a UIntPtr value.
Explicit(Int128 to UInt64)	Explicitly converts a 128-bit signed integer to a UInt64 value.
Explicit(Int128 to UInt32)	Explicitly converts a 128-bit signed integer to a UInt32 value.
Explicit(Int128 to UInt16)	Explicitly converts a 128-bit signed integer to a UInt16 value.
Explicit(Int128 to UInt128)	Explicitly converts a 128-bit signed integer to a UInt128 value.
Explicit(Int128 to Single)	Explicitly converts a 128-bit signed integer to a Single value.
Explicit(Int128 to SByte)	Explicitly converts a 128-bit signed integer to a SByte value.
Explicit(Int128 to IntPtr)	Explicitly converts a 128-bit signed integer to a IntPtr value.
Explicit(Int128 to Int32)	Explicitly converts a 128-bit signed integer to a Int32 value.
Explicit(Int128 to Int16)	Explicitly converts a 128-bit signed integer to a Int16 value.
Explicit(Int128 to Half)	Explicitly converts a 128-bit signed integer to a Half value.
Explicit(Int128 to Double)	Explicitly converts a 128-bit signed integer to a Double value.
Explicit(Int128 to Decimal)	Explicitly converts a 128-bit signed integer to a Decimal value.
Explicit(Int128 to Char)	Explicitly converts a 128-bit signed integer to a Char value.
Explicit(Int128 to Byte)	Explicitly converts a 128-bit signed integer to a Byte value.
Explicit(Double to Int128)	Explicitly converts a Double value to a 128-bit signed integer.
Explicit(Decimal to Int128)	Explicitly converts a Decimal value to a 128-bit signed integer.

Explicit(Int128 to Int64)	Explicitly converts a 128-bit signed integer to a Int64 value.

Explicit(Single to Int128)

Explicitly converts a Single value to a 128-bit signed integer.

C#

```
public static explicit operator Int128 (float value);
```

Parameters

`value` Single

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to UIntPtr)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a UIntPtr value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator UIntPtr (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

UIntPtr

`value` converted to a UIntPtr.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to UInt64)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a UInt64 value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator ulong (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

UInt64

`value` converted to a UInt64.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to UInt32)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a UInt32 value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator uint (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

UInt32

`value` converted to a UInt32.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to UInt16)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a UInt16 value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator ushort (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

UInt16

`value` converted to a UInt16.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to UInt128)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a UInt128 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator UInt128 (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

UInt128

`value` converted to a UInt128.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Single)

Explicitly converts a 128-bit signed integer to a Single value.

```
C#

public static explicit operator float (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Single

`value` converted to a Single.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to SByte)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit signed integer to a SByte value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator sbyte (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

SByte

`value` converted to a SByte.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to IntPtr)

Explicitly converts a 128-bit signed integer to a IntPtr value.

```
C#

public static explicit operator IntPtr (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

IntPtr

`value` converted to a IntPtr.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Int32)

Explicitly converts a 128-bit signed integer to a Int32 value.

```
C#
public static explicit operator int (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Int32

`value` converted to a Int32.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Int16)

Explicitly converts a 128-bit signed integer to a Int16 value.

```
C#
public static explicit operator short (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Int16

`value` converted to a Int16.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Half)

Explicitly converts a 128-bit signed integer to a Half value.

```
C#
public static explicit operator Half (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Half

`value` converted to a Half.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Double)

Explicitly converts a 128-bit signed integer to a Double value.

```
C#
public static explicit operator double (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Double

`value` converted to a Double.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Decimal)

Explicitly converts a 128-bit signed integer to a Decimal value.

```
C#
public static explicit operator decimal (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Decimal

`value` converted to a Decimal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Char)

Explicitly converts a 128-bit signed integer to a Char value.

```
C#
public static explicit operator char (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Char

`value` converted to a Char.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Byte)

Explicitly converts a 128-bit signed integer to a Byte value.

```
C#
public static explicit operator byte (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Byte

`value` converted to a Byte.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Double to Int128)

Explicitly converts a Double value to a 128-bit signed integer.

```
C#
public static explicit operator Int128 (double value);
```

Parameters

`value` Double

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Decimal to Int128)

Explicitly converts a Decimal value to a 128-bit signed integer.

```
C#
public static explicit operator Int128 (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int128 to Int64)

Explicitly converts a 128-bit signed integer to a Int64 value.

```
C#
public static explicit operator long (Int128 value);
```

Parameters

`value` Int128

The value to convert.

Returns

Int64

`value` converted to a Int64.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.GreaterThan(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#
public static bool operator > (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.GreaterThanOrEqual(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#
public static bool operator >= (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(SByte to Int128)	Implicitly converts a SByte value to a 128-bit signed integer.
Implicit(UIntPtr to Int128)	Implicitly converts a UIntPtr value to a 128-bit signed integer.
Implicit(UInt32 to Int128)	Implicitly converts a UInt32 value to a 128-bit signed integer.
Implicit(UInt16 to Int128)	Implicitly converts a UInt16 value to a 128-bit signed integer.
Implicit(IntPtr to Int128)	Implicitly converts a IntPtr value to a 128-bit signed integer.
Implicit(UInt64 to Int128)	Implicitly converts a UInt64 value to a 128-bit signed integer.
Implicit(Int32 to Int128)	Implicitly converts a Int32 value to a 128-bit signed integer.
Implicit(Int16 to Int128)	Implicitly converts a Int16 value to a 128-bit signed integer.
Implicit(Char to Int128)	Implicitly converts a Char value to a 128-bit signed integer.
Implicit(Int64 to Int128)	Implicitly converts a Int64 value to a 128-bit signed integer.
Implicit(Byte to Int128)	Implicitly converts a Byte value to a 128-bit signed integer.

Implicit(SByte to Int128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a SByte value to a 128-bit signed integer.

```
C#

[System.CLSCompliant(false)]
public static implicit operator Int128 (sbyte value);
```

Parameters

`value` SByte

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UIntPtr to Int128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UIntPtr value to a 128-bit signed integer.

```
C#

[System.CLSCompliant(false)]
public static implicit operator Int128 (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt32 to Int128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UInt32 value to a 128-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator Int128 (uint value);
```

Parameters

`value` UInt32

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt16 to Int128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UInt16 value to a 128-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator Int128 (ushort value);
```

Parameters

`value` UInt16

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(IntPtr to Int128)

Implicitly converts a IntPtr value to a 128-bit signed integer.

```
C#
public static implicit operator Int128 (IntPtr value);
```

Parameters

`value` IntPtr

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt64 to Int128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UInt64 value to a 128-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator Int128 (ulong value);
```

Parameters

`value` UInt64

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Int32 to Int128)

Implicitly converts a Int32 value to a 128-bit signed integer.

C#

```
public static implicit operator Int128 (int value);
```

Parameters

`value` Int32

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Int16 to Int128)

Implicitly converts a Int16 value to a 128-bit signed integer.

```
C#
public static implicit operator Int128 (short value);
```

Parameters

`value` Int16

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Char to Int128)

Implicitly converts a Char value to a 128-bit signed integer.

```
C#
public static implicit operator Int128 (char value);
```

Parameters

`value` Char

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Int64 to Int128)

Implicitly converts a Int64 value to a 128-bit signed integer.

```
C#
public static implicit operator Int128 (long value);
```

Parameters

`value` Int64

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Byte to Int128)

Implicitly converts a Byte value to a 128-bit signed integer.

C#

```
public static implicit operator Int128 (byte value);
```

Parameters

`value` Byte

The value to convert.

Returns

Int128

`value` converted to a 128-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Increment(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
public static Int128 operator ++ (Int128 value);
```

Parameters

`value` Int128

The value to increment.

Returns

Int128

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Inequality(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#
public static bool operator != (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.LeftShift(Int128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

C#

```
public static Int128 operator << (Int128 value, int shiftAmount);
```

Parameters

`value` Int128

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Int128

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.LessThan(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#
public static bool operator < (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.LessThanOrEqual(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#
public static bool operator <= (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value to compare with `right`.

`right` Int128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Modulus(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#
public static Int128 operator % (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value which `right` divides.

`right` Int128

The value which divides `left`.

Returns

Int128

The modulus or remainder of `left` divided-by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Multiply(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#
public static Int128 operator * (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value which `right` multiplies.

`right` Int128

The value which multiplies `left`.

Returns

Int128

The product of `left` multiplied-by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.OnesComplement(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
public static Int128 operator ~ (Int128 value);
```

Parameters

`value` Int128

The value for which to compute the ones-complement.

Returns

Int128

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.RightShift(Int128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#
public static Int128 operator >> (Int128 value, int shiftAmount);
```

Parameters

`value` Int128

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int128

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Remarks

This operation is meant to perform a signed (otherwise known as an arithmetic) right shift on signed types.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.Subtraction(Int128, Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#
public static Int128 operator - (Int128 left, Int128 right);
```

Parameters

`left` Int128

The value from which `right` is subtracted.

`right` Int128

The value which is subtracted from `left`.

Returns

Int128

The difference of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.UnaryNegation(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
public static Int128 operator - (Int128 value);
```

Parameters

`value` Int128

The value for which to compute the unary negation.

Returns

Int128

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.UnaryPlus(Int128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
public static Int128 operator + (Int128 value);
```

Parameters

`value` Int128

The value for which to compute the unary plus.

Returns

Int128

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.UnsignedRightShift(Int128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#
public static Int128 op_UnsignedRightShift (Int128 value, int shiftAmount);
```

Parameters

`value` Int128

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int128

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Remarks

This operation is meant to perform n unsigned (otherwise known as a logical) right shift on all types.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IAdditiveIdentity<Int128,Int128>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static Int128
System.Numerics.IAdditiveIdentity<System.Int128,System.Int128>.AdditiveIdentity { get; }
```

Property Value

Int128

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int128.IBinaryInteger<Int128>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<Int128>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int128.IBinaryInteger<Int128>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

```
C#
int IBinaryInteger<Int128>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int128.IBinaryInteger<Int128>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<Int128>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out Int128 value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int128.IBinaryInteger<Int128>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<Int128>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out Int128 value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int128.IBinaryInteger<Int128>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<Int128>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IBinaryInteger<Int128>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<Int128>.TryWriteLittleEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.IBinaryNumber<Int128>.AllBits Set Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static Int128 System.Numerics.IBinaryNumber<System.Int128>.AllBitsSet { get;
}
```

Property Value

Int128

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int128.IMultiplicativeIdentity<Int128,Int128>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static Int128
System.Numerics.IMultiplicativeIdentity<System.Int128,System.Int128>.MultiplicativeIdentity { get; }
```

Property Value

Int128

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int128.INumber<Int128>.MaxNumber(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static Int128 INumber<Int128>.MaxNumber (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumber<Int128>.MinNumber(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static Int128 INumber<Int128>.MinNumber (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsCanonical(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<Int128>.IsCanonical (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsComplexNumber(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<Int128>.IsComplexNumber (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsFinite(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<Int128>.IsFinite (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsImaginaryNumber(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<Int128>.IsImaginaryNumber (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsInfinity(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<Int128>.IsInfinity (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsInteger(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<Int128>.IsInteger (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsNaN(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<Int128>.IsNaN (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsNegativeInfinity(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<Int128>.IsNegativeInfinity (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsNormal(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<Int128>.IsNormal (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsPositiveInfinity(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<Int128>.IsPositiveInfinity (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsRealNumber(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<Int128>.IsRealNumber (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsSubnormal(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<Int128>.IsSubnormal (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.IsZero(Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<Int128>.IsZero (Int128 value);
```

Parameters

`value` Int128

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.MaxMagnitudeNumber(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static Int128 INumberBase<Int128>.MaxMagnitudeNumber (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.MinMagnitudeNumber(Int128, Int128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static Int128 INumberBase<Int128>.MinMagnitudeNumber (Int128 x, Int128 y);
```

Parameters

x Int128

The value to compare with `y`.

y Int128

The value to compare with `x`.

Returns

Int128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Int128>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Int128>.TryConvertFromChecked<TOther> (TOther value,
out Int128 result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int128.INumberBase<Int128>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Int128>.TryConvertFromSaturating<TOther> (TOther
value, out Int128 result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<Int128>.TryConvertFromTruncating<TOther> (TOther
value, out Int128 result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<Int128>.TryConvertToChecked<TOther> (Int128 value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int128.INumberBase<Int128>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<Int128>.TryConvertToSaturating<TOther> (Int128
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int128.INumberBase<Int128>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<Int128>.TryConvertToTruncating<TOther> (Int128
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 16-bit signed integer.

C#

```
public readonly struct Int16 : IComparable<short>, IConvertible,
IEquatable<short>, IParsable<short>, ISpanParsable<short>,
IUtf8SpanParsable<short>,
System.Numerics.IAdditionOperators<short,short,short>,
System.Numerics.IAdditiveIdentity<short,short>,
System.Numerics.IBinaryInteger<short>, System.Numerics.IBinaryNumber<short>,
System.Numerics.IBitwiseOperators<short,short,short>,
System.Numerics.IComparisonOperators<short,short,bool>,
System.Numerics.IDecrementOperators<short>,
System.Numerics.IDivisionOperators<short,short,short>,
System.Numerics.IEqualityOperators<short,short,bool>,
System.Numerics.IIncrementOperators<short>,
System.Numerics.IMinMaxValue<short>,
System.Numerics.IModulusOperators<short,short,short>,
System.Numerics.IMultiplicativeIdentity<short,short>,
System.Numerics.IMultiplyOperators<short,short,short>,
System.Numerics.INumber<short>, System.Numerics.INumberBase<short>,
System.Numerics.IShiftOperators<short,int,short>,
System.Numerics.ISignedNumber<short>,
System.Numerics.ISubtractionOperators<short,short,short>,
System.Numerics.IUnaryNegationOperators<short,short>,
System.Numerics.IUnaryPlusOperators<short,short>
```

Inheritance Object → ValueType → Int16

Implements IComparable , IComparable<Int16> , IConvertible , IEquatable<Int16> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Int16> , IParsable<TSelf> , ISpanParsable<Int16> , ISpanParsable<TSelf> , IAdditionOperators<Int16,Int16,Int16> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Int16,Int16> ,

IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Int16> ,

IBinaryNumber<Int16> , IBinaryNumber<TSelf> ,

IBitwiseOperators<Int16,Int16,Int16> ,

IBitwiseOperators<TSelf,TSelf,TSelf> ,

IComparisonOperators<Int16,Int16,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<Int16> , IDecrementOperators<TSelf> ,

IDivisionOperators<Int16,Int16,Int16> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<Int16,Int16,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Int16> ,

IIncrementOperators<TSelf> , IMinMaxValue<Int16> ,

IModulusOperators<Int16,Int16,Int16> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<Int16,Int16> , IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<Int16,Int16,Int16> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Int16> ,

INumber<TSelf> , INumberBase<Int16> , INumberBase<TSelf> ,

IShiftOperators<Int16,Int32,Int16> , IShiftOperators<TSelf,Int32,TSelf> ,

ISignedNumber<Int16> , ISubtractionOperators<Int16,Int16,Int16> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<Int16,Int16> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Int16,Int16> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Int16> ,

IUtf8SpanParsable<TSelf>

Remarks

The Int16 value type represents signed integers with values ranging from negative 32768 through positive 32767.

This type provides methods to convert the value of an instance of this type to its string representation, to convert the string representation of a number to an instance of this type, and to compare instances of this type.

You can also call the members of the Math class to perform a wide range of numeric operations, including getting the absolute value of a number, calculating the quotient and remainder from integral division, determining the maximum or minimum value of two integers, getting the sign of a number, and rounding a number.

For information about how format specification codes control the string representation of value types, see Formatting Types.

Fields

Expand table

MaxValue	Represents the largest possible value of an Int16. This field is constant.
MinValue	Represents the smallest possible value of Int16. This field is constant.

Methods

Expand table

Abs(Int16)	Computes the absolute of a value.
Clamp(Int16, Int16, Int16)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Int16)	Compares this instance to a specified 16-bit signed integer and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified 16-bit signed integer.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the object.
CopySign(Int16, Int16)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(Int16, Int16)	Computes the quotient and remainder of two values.
Equals(Int16)	Returns a value indicating whether this instance is equal to a specified Int16 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Int16.
IsEvenInteger(Int16)	Determines if a value represents an even integral number.
IsNegative(Int16)	Determines if a value is negative.
IsOddInteger(Int16)	Determines if a value represents an odd integral number.
IsPositive(Int16)	Determines if a value is positive.
IsPow2(Int16)	Determines if a value is a power of two.
LeadingZeroCount(Int16)	Computes the number of leading zeros in a value.
Log2(Int16)	Computes the log2 of a value.
Max(Int16, Int16)	Compares two values to compute which is greater.
MaxMagnitude(Int16, Int16)	Compares two values to compute which is greater.
Min(Int16, Int16)	Compares two values to compute which is lesser.
MinMagnitude(Int16, Int16)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer

	equivalent.
Parse(String)	Converts the string representation of a number to its 16-bit signed integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 16-bit signed integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 16-bit signed integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent.
PopCount(Int16)	Computes the number of bits that are set in a value.
RotateLeft(Int16, Int32)	Rotates a value left by a given amount.
RotateRight(Int16, Int32)	Rotates a value right by a given amount.
Sign(Int16)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific formatting information.
TrailingZeroCount(Int16)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current short number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int16)	Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit signed integer

	equivalent.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int16)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, Int16)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int16)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Int16)	Tries to parse a string into a value.
TryParse(String, Int16)	Converts the string representation of a number to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Int16)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<Int16,Int16,Int16>.Addition(Int16, Int16)	Adds two values together to compute their sum.
IAdditionOperators<Int16,Int16,Int16>.CheckedAddition(Int16, Int16)	Adds two values together to compute their sum.
IAdditiveIdentity<Int16,Int16>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<Int16>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

IBinaryInteger<Int16>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Int16>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Int16)	
IBinaryInteger<Int16>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Int16)	
IBinaryInteger<Int16>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Int16>.TryWriteLittle Endian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Int16>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Int16,Int16,Int16>.BitwiseAnd(Int16, Int16)	Computes the bitwise-and of two values.
IBitwiseOperators<Int16,Int16,Int16>.BitwiseOr(Int16, Int16)	Computes the bitwise-or of two values.
IBitwiseOperators<Int16,Int16,Int16>.ExclusiveOr(Int16, Int16)	Computes the exclusive-or of two values.
IBitwiseOperators<Int16,Int16,Int16>.Ones Complement(Int16)	Computes the ones-complement representation of a given value.
IComparisonOperators<Int16,Int16,Boolean>.Greater Than(Int16, Int16)	Compares two values to determine which is greater.
IComparisonOperators<Int16,Int16,Boolean>.Greater ThanOrEqual(Int16, Int16)	Compares two values to determine which is greater or equal.
IComparisonOperators<Int16,Int16,Boolean>.Less Than(Int16, Int16)	Compares two values to determine which is less.
IComparisonOperators<Int16,Int16,Boolean>.LessThan OrEqual(Int16, Int16)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws

	an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Int16>.Checked Decrement(Int16)	Decrements a value.
IDecrementOperators<Int16>.Decrement(Int16)	Decrements a value.
IDivisionOperators<Int16,Int16,Int16>.Division(Int16, Int16)	Divides one value by another to compute their quotient.
IEquality Operators<Int16,Int16,Boolean>.Equality(Int16, Int16)	Compares two values to determine equality.
IEquality Operators<Int16,Int16,Boolean>.Inequality(Int16, Int16)	Compares two values to determine inequality.
IIncrementOperators<Int16>.CheckedIncrement(Int16)	Increments a value.
IIncrementOperators<Int16>.Increment(Int16)	Increments a value.

IMinMaxValue<Int16>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Int16>.MinValue	Gets the minimum value of the current type.
IModulusOperators<Int16,Int16,Int16>.Modulus(Int16, Int16)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Int16,Int16>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
IMultiplyOperators<Int16,Int16,Int16>.CheckedMultiply(Int16, Int16)	Multiplies two values together to compute their product.
IMultiplyOperators<Int16,Int16,Int16>.Multiply(Int16, Int16)	Multiplies two values together to compute their product.
INumber<Int16>.MaxNumber(Int16, Int16)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<Int16>.MinNumber(Int16, Int16)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Int16>.IsCanonical(Int16)	Determines if a value is in its canonical representation.
INumberBase<Int16>.IsComplexNumber(Int16)	Determines if a value represents a complex number.
INumberBase<Int16>.IsFinite(Int16)	Determines if a value is finite.
INumberBase<Int16>.IsImaginaryNumber(Int16)	Determines if a value represents a pure imaginary number.
INumberBase<Int16>.IsInfinity(Int16)	Determines if a value is infinite.
INumberBase<Int16>.IsInteger(Int16)	Determines if a value represents an integral number.
INumberBase<Int16>.IsNaN(Int16)	Determines if a value is NaN.
INumberBase<Int16>.IsNegativeInfinity(Int16)	Determines if a value is negative infinity.
INumberBase<Int16>.IsNormal(Int16)	Determines if a value is normal.
INumberBase<Int16>.IsPositiveInfinity(Int16)	Determines if a value is positive infinity.

INumberBase<Int16>.IsRealNumber(Int16)	Determines if a value represents a real number.
INumberBase<Int16>.IsSubnormal(Int16)	Determines if a value is subnormal.
INumberBase<Int16>.IsZero(Int16)	Determines if a value is zero.
INumberBase<Int16>.MaxMagnitudeNumber(Int16, Int16)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Int16>.MinMagnitudeNumber(Int16, Int16)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Int16>.One	Gets the value `1` for the type.
INumberBase<Int16>.Radix	Gets the radix, or base, for the type.
INumberBase<Int16>.TryConvertFromChecked<TOther>(TOther, Int16)	
INumberBase<Int16>.TryConvertFromSaturating<TOther>(TOther, Int16)	
INumberBase<Int16>.TryConvertFromTruncating<TOther>(TOther, Int16)	
INumberBase<Int16>.TryConvertToChecked<TOther>(Int16, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Int16>.TryConvertToSaturating<TOther>(Int16, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Int16>.TryConvertToTruncating<TOther>(Int16, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Int16>.Zero	Gets the value `0` for the type.
IShiftOperators<Int16,Int32,Int16>.LeftShift(Int16, Int32)	Shifts a value left by a given amount.
IShiftOperators<Int16,Int32,Int16>.RightShift(Int16, Int32)	Shifts a value right by a given amount.
IShiftOperators<Int16,Int32,Int16>.UnsignedRightShift(Int16, Int32)	Shifts a value right by a given amount.

ISignedNumber<Int16>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<Int16,Int16,Int16>.Checked Subtraction(Int16, Int16)	Subtracts two values to compute their difference.
ISubtraction Operators<Int16,Int16,Int16>.Subtraction(Int16, Int16)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Int16,Int16>.CheckedUnary Negation(Int16)	Computes the checked unary negation of a value.
IUnaryNegationOperators<Int16,Int16>.Unary Negation(Int16)	Computes the unary negation of a value.
IUnaryPlusOperators<Int16,Int16>.UnaryPlus(Int16)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- UInt16
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of an Int16. This field is constant.

C#

```
public const short MaxValue = 32767;
```

Field Value

Value* = *32767

Int16

Examples

The following example uses the MaxValue property to prevent an OverflowException when converting to an Int16 value.

C#

```
long[] numbersToConvert = {162345, 32183, -54000};
short newNumber;
foreach (long number in numbersToConvert)
{
   if (number >= Int16.MinValue && number <= Int16.MaxValue)
   {
      newNumber = Convert.ToInt16(number);
      Console.WriteLine($"Successfully converted {newNumber} to an Int16.");
   }
   else
   {
      Console.WriteLine($"Unable to convert {number} to an Int16.");
   }
}
// The example displays the following output to the console:
//       Unable to convert 162345 to an Int16.
```

```
//       Successfully converted 32183 to an Int16.
//       Unable to convert -54000 to an Int16.
```

Remarks

The value of this constant is 32767; that is, hexadecimal 0x7FFF.

The MaxValue property is typically used to prevent an OverflowException when converting from a numeric type with a greater upper range (such as a UInt16 or a Int32) to an Int16. The example illustrates this usage.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of Int16. This field is constant.

C#

```
public const short MinValue = -32768;
```

Field Value

***Value* = -32768**

Int16

Examples

The following example uses the MinValue property to prevent an OverflowException when converting to an Int16 value.

C#

```
long[] numbersToConvert = {162345, 32183, -54000};
short newNumber;
foreach (long number in numbersToConvert)
{
   if (number >= Int16.MinValue && number <= Int16.MaxValue)
   {
      newNumber = Convert.ToInt16(number);
      Console.WriteLine($"Successfully converted {newNumber} to an Int16.");
   }
   else
   {
      Console.WriteLine($"Unable to convert {number} to an Int16.");
   }
}
// The example displays the following output to the console:
//       Unable to convert 162345 to an Int16.
```

```
//       Successfully converted 32183 to an Int16.
//       Unable to convert -54000 to an Int16.
```

Remarks

The value of this constant is -32768; that is, hexadecimal 0x8000.

The MinValue property is typically used to prevent an OverflowException when converting from a numeric type with a greater lower range (such as an Int32 or an Int64) to an Int16. The example illustrates this usage.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.Abs(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static short Abs (short value);
```

Parameters

`value` Int16

The value for which to get its absolute.

Returns

Int16

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Clamp(Int16, Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

C#

```
public static short Clamp (short value, short min, short max);
```

Parameters

`value` Int16

The value to clamp.

`min` Int16

The inclusive minimum to which `value` should clamp.

`max` Int16

The inclusive maximum to which `value` should clamp.

Returns

Int16

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or another Int16 instance and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object or the other Int16 instance.

Overloads

Expand table

CompareTo(Int16)	Compares this instance to a specified 16-bit signed integer and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified 16-bit signed integer.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the object.

CompareTo(Int16)

Compares this instance to a specified 16-bit signed integer and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified 16-bit signed integer.

C#

```
public int CompareTo (short value);
```

Parameters

`value` Int16

An integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
```

```
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
```

```
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
```

```
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Int16.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- UInt16
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not an Int16.

Remarks

An Int16, regardless of its value, is considered greater than a null reference.

The `value` parameter must be `null` or an instance of Int16; otherwise, an exception is thrown.

See also

- UInt16

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.CopySign(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
public static short CopySign (short value, short sign);
```

Parameters

value Int16

The value whose magnitude is used in the result.

sign Int16

The value whose sign is used in the result.

Returns

Int16

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static short CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int16

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static short CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int16

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.CreateTruncating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static short CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int16

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.DivRem(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (short Quotient, short Remainder) DivRem (short left, short
right);
```

Parameters

`left` Int16

The value which `right` divides.

`right` Int16

The value which divides `left`.

Returns

ValueTuple<Int16,Int16>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or Int16.

Overloads

Expand table

Equals(Int16)	Returns a value indicating whether this instance is equal to a specified Int16 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(Int16)

Returns a value indicating whether this instance is equal to a specified Int16 value.

C#

```
public bool Equals (short obj);
```

Parameters

obj Int16

An Int16 value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int16) method overloads. If an implicit conversion between the `obj` argument and an Int16 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int16) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int16 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte and SByte values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int16) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static short value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,17}", value.Equals(int1));
      TestObjectForEquality(int1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
      Console.WriteLine("value = ushort1: {0,15}",
```

```
value.Equals(ushort1));
      TestObjectForEquality(ushort1);

      decimal dec1 = 112m;
      Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
      TestObjectForEquality(dec1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }

   private static void TestObjectForEquality(Object obj)
   {
      Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                        value, value.GetType().Name,
                        obj, obj.GetType().Name,
                        value.Equals(obj));
   }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int16) = 112 (Byte): False
//
//       value = int1:             False
//       112 (Int16) = 112 (Int32): False
//
//       value = sbyte1:            True
//       112 (Int16) = 112 (SByte): False
//
//       value = ushort1:            False
//       112 (Int16) = 112 (UInt16): False
//
//       value = dec1:                False
//       112 (Int16) = 112 (Decimal): False
//
//       value = dbl1:               False
//       112 (Int16) = 112 (Double): False
```

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare to this instance.

Returns

Boolean

`true` if `obj` is an instance of Int16 and equals the value of this instance; otherwise, `false`.

Examples

The following code example illustrates the use of `Equals` in the context of `Int16`, comparing two short values and returning `true` if they represent the same number, or `false` if they do not.

```
C#

Int16 myVariable1 = 20;
Int16 myVariable2 = 20;

// Get and display the declaring type.
```

```
Console.WriteLine("\nType of 'myVariable1' is '{0}' and"+
     " value is :{1}",myVariable1.GetType(), myVariable1);
Console.WriteLine("Type of 'myVariable2' is '{0}' and"+
     " value is :{1}",myVariable2.GetType(), myVariable2);

// Compare 'myVariable1' instance with 'myVariable2' Object.
if( myVariable1.Equals( myVariable2 ) )
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are equal");
else
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are not equal");
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int16) method overloads. If an implicit conversion between the `obj` argument and an Int16 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int16) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int16 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte and SByte values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int16) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static short value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,17}", value.Equals(int1));
      TestObjectForEquality(int1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);
```

```
        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,15}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int16) = 112 (Byte): False
//
//       value = int1:             False
//       112 (Int16) = 112 (Int32): False
//
//       value = sbyte1:            True
//       112 (Int16) = 112 (SByte): False
//
//       value = ushort1:            False
//       112 (Int16) = 112 (UInt16): False
//
//       value = dec1:                False
//       112 (Int16) = 112 (Decimal): False
//
//       value = dbl1:               False
//       112 (Int16) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int16.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Int16.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Int16.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IsEvenInteger(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IsNegative(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IsOddInteger(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IsPositive(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IsPow2(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.LeadingZeroCount(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static short LeadingZeroCount (short value);
```

Parameters

`value` Int16

The value whose leading zeroes are to be counted.

Returns

Int16

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Log2(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

```
C#
public static short Log2 (short value);
```

Parameters

`value` Int16

The value whose log2 is to be computed.

Returns

Int16

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Max(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static short Max (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.MaxMagnitude(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static short MaxMagnitude (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Min(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static short Min (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.MinMagnitude(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static short MinMagnitude (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 16-bit signed integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 16-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 16-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 16-bit signed integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent.

```
C#

public static short Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

Returns

Int16

A 16-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int16.MinValue or greater than Int16.MaxValue.

-or-

`s` includes non-zero fractional digits.

Examples

The following example uses a variety of `style` and `provider` parameters to parse the string representations of Int16 values.

C#

```
string value;
short number;
NumberStyles style;
CultureInfo provider;

// Parse string using "." as the thousands separator
// and " " as the decimal separator.
value = "19 694,00";
style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowThousands;
provider = new CultureInfo("fr-FR");

number = Int16.Parse(value, style, provider);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '19 694,00' converted to 19694.

try
{
   number = Int16.Parse(value, style, CultureInfo.InvariantCulture);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '19 694,00'.

// Parse string using "$" as the currency symbol for en_GB and
// en-US cultures.
value = "$6,032.00";
style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
provider = new CultureInfo("en-GB");
```

```
try
{
   number = Int16.Parse(value, style, CultureInfo.InvariantCulture);
   Console.WriteLine("'{0}' converted to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", value);
}
// Displays:
//    Unable to parse '$6,032.00'.

provider = new CultureInfo("en-US");
number = Int16.Parse(value, style, provider);
Console.WriteLine("'{0}' converted to {1}.", value, number);
// Displays:
//    '$6,032.00' converted to 6032.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern and NumberFormatInfo.CurrencyNegativePattern property of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.

Element	Description
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
,	A culture-specific thousands separator symbol. The current culture's thousands separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	A sequence of the 0 digit. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. If any digit other than 0 appears in *fractional_digits*, the method throws an OverflowException.
e	The 'e' or 'E' character, which indicates that `s` can be represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag. However, the `s` parameter must represent a number in the range of the Int16 data type and cannot have a non-zero fractional component.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with *digits* only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The , element.
NumberStyles.AllowCurrencySymbol	The *$* element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be the string representation of a hexadecimal value without a prefix. For example, "9AF3" parses successfully, but "0x9AF3" does not.. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation whose GetFormat method obtains a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- NumberStyles
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent.

C#

```
public static short Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

Returns

Int16

A 16-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static short Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int16

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its 16-bit signed integer equivalent.

```
C#

public static short Parse (string s, IFormatProvider? provider);
```

Parameters

s String

A string containing a number to convert.

provider IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

Returns

Int16

A 16-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example parses string representations of Int16 values with the Int16.Parse(String, IFormatProvider) method.

C#

```
string stringToConvert;
short number;

stringToConvert = " 214 ";
try
{
   number = Int16.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert);
}
catch (OverflowException)
{
   Console.WriteLine("'{0'} is out of range of the Int16 data type.",
                     stringToConvert);
}

stringToConvert = " + 214";
try
{
   number = Int16.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert);
```

```
}
catch (OverflowException)
{
   Console.WriteLine("'{0'} is out of range of the Int16 data type.",
                     stringToConvert);
}

stringToConvert = " +214 ";
try
{
   number = Int16.Parse(stringToConvert, CultureInfo.InvariantCulture);
   Console.WriteLine("Converted '{0}' to {1}.", stringToConvert, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to parse '{0}'.", stringToConvert);
}
catch (OverflowException)
{
   Console.WriteLine("'{0'} is out of range of the Int16 data type.",
                     stringToConvert);
}
// The example displays the following output to the console:
//       Converted ' 214 ' to 214.
//       Unable to parse ' + 214'.
//       Converted ' +214 ' to 214.
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	An optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to decimal digits, only leading and trailing spaces together with a leading sign are allowed in `s`. To explicitly define the style elements together with the culture-specific

formatting information that can be present in `s`, use the Int16.Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation that obtains a NumberFormatInfo object. The NumberFormatInfo provides culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

Converts the string representation of a number to its 16-bit signed integer equivalent.

C#

```
public static short Parse (string s);
```

Parameters

s String

A string containing a number to convert.

Returns

Int16

A 16-bit signed integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Int16.MinValue or greater than Int16.MaxValue.

Examples

The following example demonstrates how to convert a string value into a 16-bit signed integer value using the Int16.Parse(String) method. The resulting integer value is then displayed to the console.

C#

```
string value;
short number;

value = " 12603 ";
try
{
   number = Int16.Parse(value);
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to convert '{0}' to a 16-bit signed
integer.",
                     value);
}

value = " 16,054";
try
{
   number = Int16.Parse(value);
```

```
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to convert '{0}' to a 16-bit signed
integer.",
                     value);
}

value = " -17264";
try
{
   number = Int16.Parse(value);
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
}
catch (FormatException)
{
   Console.WriteLine("Unable to convert '{0}' to a 16-bit signed
integer.",
                     value);
}
// The example displays the following output to the console:
//       Converted ' 12603 ' to 12603.
//       Unable to convert ' 16,054' to a 16-bit signed integer.
//       Converted ' -17264' to -17264.
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the integer value's decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements that can be present in `s`, use either the Int16.Parse(String, NumberStyles) or the Parse method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other culture, use the Int16.Parse(String, IFormatProvider) or the Int16.Parse(String, NumberStyles, IFormatProvider) method.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static short Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int16

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static short Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int16

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its 16-bit signed integer equivalent.

```
C#

public static short Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

Returns

Int16

A 16-bit signed integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int16.MinValue or greater than Int16.MaxValue.

-or-

`s` includes non-zero fractional digits.

Examples

The following example uses the Int16.Parse(String, NumberStyles) method to parse the string representations of Int16 values using the en-US culture.

C#

```
using System;
using System.Globalization;

public class ParseSample
{
   public static void Main()
   {
      string value;
      NumberStyles style;

      // Parse a number with a thousands separator (throws an exception).
      value = "14,644";
      style = NumberStyles.None;
      ParseToInt16(value, style);

      style = NumberStyles.AllowThousands;
      ParseToInt16(value, style);

      // Parse a number with a thousands separator and decimal point.
      value = "14,644.00";
      style = NumberStyles.AllowThousands | NumberStyles.Integer |
              NumberStyles.AllowDecimalPoint;
      ParseToInt16(value, style);
```

```
      // Parse a number with a fractional component (throws an
exception).
      value = "14,644.001";
      ParseToInt16(value, style);

      // Parse a number in exponential notation.
      value = "145E02";
      style = style | NumberStyles.AllowExponent;
      ParseToInt16(value, style);

      // Parse a number in exponential notation with a positive sign.
      value = "145E+02";
      ParseToInt16(value, style);

      // Parse a number in exponential notation with a negative sign
      // (throws an exception).
      value = "145E-02";
      ParseToInt16(value, style);
   }

   private static void ParseToInt16(string value, NumberStyles style)
   {
      try
      {
         short number = Int16.Parse(value, style);
         Console.WriteLine("Converted '{0}' to {1}.", value, number);
      }
      catch (FormatException)
      {
         Console.WriteLine("Unable to parse '{0}' with style {1}.",
value,
                           style.ToString());
      }
      catch (OverflowException)
      {
         Console.WriteLine("'{0}' is out of range of the Int16 type.",
value);
      }
   }
}
// The example displays the following output to the console:
//       Unable to parse '14,644' with style None.
//       Converted '14,644' to 14644.
//       Converted '14,644.00' to 14644.
//       '14,644.001' is out of range of the Int16 type.
//       Converted '145E02' to 14500.
//       Converted '145E+02' to 14500.
//       '145E-02' is out of range of the Int16 type.
```

Remarks

The `style` parameter defines the style elements (such as white space or a sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern and NumberFormatInfo.CurrencyNegativePattern property of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
,	A culture-specific thousands separator symbol. The current culture's thousands separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	A sequence of the 0 digit. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. If any digit other than 0 appears in *fractional_digits*, the method throws an OverflowException.

Element	Description
e	The 'e' or 'E' character, which indicates that `s` can be represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag. However, the `s` parameter must represent a number in the range of the Int16 data type and cannot have a non-zero fractional component.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with *digits* only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The , element.

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.AllowCurrencySymbol	The $ element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be the string representation of a hexadecimal value without a prefix. For example, "9AF3" parses successfully, but "0x9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse `s` using the formatting information of a specific culture, call the Int16.Parse(String, NumberStyles, IFormatProvider) method.

See also

- NumberStyles
- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.PopCount(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static short PopCount (short value);
```

Parameters

`value` Int16

The value whose set bits are to be counted.

Returns

Int16

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.RotateLeft(Int16, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static short RotateLeft (short value, int rotateAmount);
```

Parameters

`value` Int16

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Int16

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.RotateRight(Int16, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static short RotateRight (short value, int rotateAmount);
```

Parameters

`value` Int16

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Int16

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.Sign(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (short value);
```

Parameters

`value` Int16

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific formatting information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a minus sign if the value is negative, and a sequence of digits ranging from 0 to 9 with no leading zeroes.

Examples

The following example uses the ToString() method to display an array of Int16 values to the console.

C#

```
short[] numbers = {0, 14624, 13982, short.MaxValue,
                   short.MinValue, -16667};
foreach (short number in numbers)
{
   Console.WriteLine(number.ToString());
}
// The example displays the following output to the console:
//       0
//       14624
//       13982
//       32767
//       -32768
//       -16667
```

Remarks

The ToString() method formats an Int16 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example iterates an array of Int16 values and displays each of them to the console by calling the Int16.ToString(IFormatProvider) method with different format providers. Because of the simple formatting defined by the default "G" format specifier, the formatted strings produced for each Int16 value are identical regardless of the value of the `provider` parameter.

C#

```
short[] numbers = {-23092, 0, 14894, Int16.MaxValue};
CultureInfo[] providers = {new CultureInfo("en-us"),
                           new CultureInfo("fr-fr"),
                           new CultureInfo("de-de"),
                           new CultureInfo("es-es")};
foreach (Int16 int16Value in numbers)
{
   foreach (CultureInfo provider in providers)
   {
      Console.Write("{0, 6} ({1})     ",
                    int16Value.ToString(provider),
                    provider.Name);
   }
   Console.WriteLine();
}
// The example displays the following output to the console:
//       -23092 (en-US)     -23092 (fr-FR)     -23092 (de-DE)     -23092
(es-ES)
//            0 (en-US)          0 (fr-FR)          0 (de-DE)          0
(es-ES)
//        14894 (en-US)      14894 (fr-FR)      14894 (de-DE)      14894
(es-ES)
//        32767 (en-US)      32767 (fr-FR)      32767 (de-DE)      32767
(es-ES)
```

Remarks

The ToString(IFormatProvider) method formats an Int16 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you

want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose IFormatProvider.GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a NumberFormatInfo object or a CultureInfo object. The NumberFormatInfo object provides culture-specific information about the format of the string returned by this method. If `provider` is `null`, this instance is formatted with the NumberFormatInfo object for the current culture.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Examples

The following example initializes two Int16 values and displays them to the console using each of the supported standard format strings and several custom format strings. The example is run with en-US as the current culture.

C#

```
Int16[] values = {-23805, 32194};
string[] formats = {"C4", "D6", "e1", "E2", "F1", "G", "N1",
                    "P0", "X4", "000000.0000", "##000.0"};
foreach (string format in formats)
{
   Console.WriteLine("'{0,2}' format specifier: {1,17}   {2,17}",
                     format,
                     values[0].ToString(format),
                     values[1].ToString(format));
```

```
}
// The example displays the following output to the console:
//    'C4' format specifier:    ($23,805.0000)        $32,194.0000
//    'D6' format specifier:           -023805              032194
//    'e1' format specifier:         -2.4e+004            3.2e+004
//    'E2' format specifier:        -2.38E+004           3.22E+004
//    'F1' format specifier:          -23805.0             32194.0
//    ' G' format specifier:            -23805               32194
//    'N1' format specifier:         -23,805.0            32,194.0
//    'P0' format specifier:      -2,380,500 %         3,219,400 %
//    'X4' format specifier:              A303                 7DC2
//    '000000.0000' format specifier:      -023805.0000
032194.0000
//    '##000.0' format specifier:          -23805.0             32194.0
```

Remarks

The ToString(String) method formats an Int16 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string, the return value of this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The return value of this instance is formatted with the NumberFormatInfo for the current culture.

See also

- Parse(String)
- String
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific formatting information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Examples

The following example displays an Int16 value using each of the supported standard format strings in four different cultures.

C#

```
Int16 value = 14603;
string[] formats = {"C", "D6", "e1", "E2", "F1", "G", "N1",
                    "P0", "X4", "000000.0000", "##000.0"};
CultureInfo[] providers = {new CultureInfo("en-us"),
                           new CultureInfo("fr-fr"),
                           new CultureInfo("de-de"),
                           new CultureInfo("es-es")};
// Display header.
Console.WriteLine("{0,24}{1,14}{2,14}{3,14}", providers[0], providers[1],
                  providers[2], providers[3]);
Console.WriteLine();
// Display a value using each format string.
foreach (string format in formats)
{
   // Display the value for each provider on the same line.
   Console.Write("{0,-12}", format);
   foreach (CultureInfo provider in providers)
   {
      Console.Write("{0,12}  ",
                    value.ToString(format, provider));
   }
   Console.WriteLine();
}
// The example displays the following output to the console:
//                        en-US         fr-FR         de-DE         es-ES
//
//    C             $14,603.00   14 603,00 €   14.603,00 €   14.603,00 €
//    D6                014603        014603        014603        014603
//    e1              1.5e+004      1,5e+004      1,5e+004      1,5e+004
//    E2             1.46E+004     1,46E+004     1,46E+004     1,46E+004
//    F1               14603.0       14603,0       14603,0       14603,0
//    G                  14603         14603         14603         14603
```

```
//    N1              14,603.0      14 603,0      14.603,0      14.603,0
//    P0           1,460,300 %   1 460 300 %    1.460.300%   1.460.300 %
//    X4                  390B          390B          390B          390B
//    000000.0000  014603.0000   014603,0000   014603,0000   014603,0000
//    ##000.0          14603.0       14603,0       14603,0       14603,0
```

Remarks

The ToString(String, IFormatProvider) method formats an Int16 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string (""), the string returned by this method is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.

- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider.

If `provider` is `null`, or a NumberFormatInfo object cannot be obtained from `provider`, the return value is formatted with the NumberFormatInfo for the current culture.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Int16.TrailingZeroCount(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static short TrailingZeroCount (short value);
```

Parameters

`value` Int16

The value whose trailing zeroes are to be counted.

Returns

Int16

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current short number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current short number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Int16)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Int16)	Converts the string representation of a number to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int16)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Int16)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int16)	Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit signed integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int16)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(String, NumberStyles, IFormatProvider, Int16)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int16)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out short result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int16

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Int16)

Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out short result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert.

`result` Int16

When this method returns, contains the 16-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int16.MinValue or greater than Int16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Int16)

Converts the string representation of a number to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

```C#
public static bool TryParse (string? s, out short result);
```

Parameters

`s` String

A string containing a number to convert.

`result` Int16

When this method returns, contains the 16-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Int16.MinValue or greater than Int16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the Int16.TryParse(String, Int16) method with a number of different string values.

```C#
using System;

public class StringParsing
{
   public static void Main()
   {
      TryToParse(null);
      TryToParse("16051");
      TryToParse("9432.0");
      TryToParse("16,667");
      TryToParse("   -322   ");
```

```
        TryToParse("+4302");
        TryToParse("(100);");
        TryToParse("01FA");
    }

    private static void TryToParse(string value)
    {
        short number;
        bool result = Int16.TryParse(value, out number);
        if (result)
        {
            Console.WriteLine("Converted '{0}' to {1}.", value, number);
        }
        else
        {
            if (value == null) value = "";
            Console.WriteLine("Attempted conversion of '{0}' failed.",
value);
        }
    }
}
// The example displays the following output to the console:
//       Attempted conversion of '' failed.
//       Converted '16051' to 16051.
//       Attempted conversion of '9432.0' failed.
//       Attempted conversion of '16,667' failed.
//       Converted '   -322   ' to -322.
//       Converted '+4302' to 4302.
//       Attempted conversion of '(100)' failed.
//       Attempted conversion of '01FA' failed.
```

Some of the strings that the TryParse(String, Int16) method is unable to convert in this example are:

- "9432.0". The conversion fails because the string cannot contain a decimal separator; it must contain integral digits only.
- "16,667". The conversion fails because the string cannot contain group separators; it must contain integral digits only.
- "(100)". The conversion fails because the string cannot contain a negative sign other than the one defined by the current culture's NumberFormatInfo.NegativeSign and NumberFormatInfo.NumberNegativePattern properties.
- "01FA". The conversion fails because the string cannot contain hexadecimal digits; it must contain decimal digits only.

Remarks

The Int16.TryParse(String, Int16) method differs from the Int16.Parse(String) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed Int16 value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `s` parameter should be the string representation of a number in the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Int16.TryParse(String, NumberStyles, IFormatProvider, Int16) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see CurrentInfo.

This overload of the TryParse method interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the Int16.TryParse(String, NumberStyles, IFormatProvider, Int16) overload.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, Int16)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out short result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int16

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Int16)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider, out
short result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int16

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int16)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
short result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int16

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Int16)

Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit signed integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out short
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Int16

When this method returns, contains the 16-bit signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int16)

Converts the span representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
short result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert. The span is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Int16

When this method returns, contains the 16-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int16.MinValue or greater than Int16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Int16)

Converts the string representation of a number in a specified style and culture-specific format to its 16-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out short result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Int16

When this method returns, contains the 16-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int16.MinValue or greater than

Int16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the Int16.TryParse(String, NumberStyles, IFormatProvider, Int16) method with a number of different string values.

C#

```
using System;
using System.Globalization;

public class StringParsing
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "10677";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-30677";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);

      numericString = "10345.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
```

```
        styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
        CallTryParse(numericString, styles);

        numericString = "22,593";
        styles = NumberStyles.Integer | NumberStyles.AllowThousands;
        CallTryParse(numericString, styles);

        numericString = "12E-01";
        styles = NumberStyles.Integer | NumberStyles.AllowExponent;
        CallTryParse(numericString, styles);

        numericString = "12E03";
        CallTryParse(numericString, styles);

        numericString = "80c1";
        CallTryParse(numericString, NumberStyles.HexNumber);

        numericString = "0x80C1";
        CallTryParse(numericString, NumberStyles.HexNumber);
    }

    private static void CallTryParse(string stringToConvert, NumberStyles
styles)
    {
        short number;
        bool result = Int16.TryParse(stringToConvert, styles,
                                     CultureInfo.InvariantCulture, out
number);
        if (result)
            Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
        else
            Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
    }
}
// The example displays the following output to the console:
//       Converted '10677' to 10677.
//       Attempted conversion of '-30677' failed.
//       Converted '10345.00' to 10345.
//       Attempted conversion of '10345.72' failed.
//       Converted '22,593' to 22593.
//       Attempted conversion of '12E-01' failed.
//       Converted '12E03' to 12000.
//       Converted '80c1' to -32575.
//       Attempted conversion of '0x80C1' failed.
```

Remarks

The Int16.TryParse(String, NumberStyles, IFormatProvider, Int16) method differs from the Int16.Parse(String, NumberStyles, IFormatProvider) method by returning a Boolean value that indicates whether the parse operation succeeded instead of

returning the parsed Int16 value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of style, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]digits][ws]

Or, if the `style` parameter includes NumberStyles.AllowHexSpecifier:

[ws]hexdigits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign.
digits	A sequence of digits from 0 through 9.
,	A culture-specific thousands separator. The thousands separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.

Element	Description
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The . and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or the method returns `false`.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation. If `s` represents a number in exponential notation, it must represent an integer within the range of the Int16 data type without a non-zero fractional component.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of *s*.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of *s*.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing

Non-composite NumberStyles values	Elements permitted in s in addition to digits
	the numeric value.
NumberStyles.AllowThousands	The , element.
NumberStyles.AllowCurrencySymbol	The *$* element.
NumberStyles.Currency	All elements. The *s* parameter cannot represent a hexadecimal number or a number in exponential notation.
NumberStyles.Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the . symbol. The `s` parameter can also use exponential notation.
NumberStyles.Number	The *ws*, *sign*, thousands separator (,), and decimal point (.) elements.
NumberStyles.Any	All styles, except `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "9AF3" parses successfully, but "0x9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a CultureInfo object or a NumberFormatInfo object, whose GetFormat method returns a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse
- NumberStyles
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Int16) method

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

```
C#
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

```
C#
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, unchanged.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.To Int64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToSByte(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The type to which to convert this Int16 value.

`provider` IFormatProvider

An IFormatProvider implementation that supplies information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, unchanged.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt16(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt32(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int16 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt64(Int16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.IAdditionOperators<Int16,Int16,Int16>.AdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static short IAdditionOperators<short,short,short>.operator + (short left,
short right);
```

Parameters

left Int16

The value to which `right` is added.

right Int16

The value that's added to `left`.

Returns

Int16

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IAdditionOperators<Int16,Int16,Int16>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static short IAdditionOperators<short,short,short>.op_CheckedAddition (short
left, short right);
```

Parameters

`left` Int16

The value to which `right` is added.

`right` Int16

The value that's added to `left`.

Returns

Int16

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IAdditiveIdentity<Int16,Int16>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static short
System.Numerics.IAdditiveIdentity<System.Int16,System.Int16>.AdditiveIdentity { get; }
```

Property Value

Int16

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.IBinaryInteger<Int16>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<short>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.IBinaryInteger<Int16>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<short>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int16.IBinaryInteger<Int16>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<short>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out short value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int16.IBinaryInteger<Int16>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<short>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out short value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int16.IBinaryInteger<Int16>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<short>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBinaryInteger<Int16>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<short>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBinaryNumber<Int16>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static short System.Numerics.IBinaryNumber<System.Int16>.AllBitsSet { get; }
```

Property Value

Int16

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBitwiseOperators<Int16,Int16,Int16>.BitwiseAndOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static short IBitwiseOperators<short,short,short>.operator & (short left,
short right);
```

Parameters

`left` Int16

The value to `and` with `right`.

`right` Int16

The value to `and` with `left`.

Returns

Int16

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBitwiseOperators<Int16,Int16,Int16>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static short IBitwiseOperators<short,short,short>.operator | (short left,
short right);
```

Parameters

`left` Int16

The value to `or` with `right`.

`right` Int16

The value to `or` with `left`.

Returns

Int16

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBitwiseOperators<Int16,Int16,Int16>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static short IBitwiseOperators<short,short,short>.operator ^ (short left,
short right);
```

Parameters

left Int16

The value to xor with `right`.

right Int16

The value to xor with `left`.

Returns

Int16

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IBitwiseOperators<Int16,Int16,Int16>.OnesComplement(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static short IBitwiseOperators<short,short,short>.operator ~ (short value);
```

Parameters

`value` Int16

The value for which to compute the ones-complement.

Returns

Int16

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IComparisonOperators<Int16,Int16,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<short,short,bool>.operator > (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IComparisonOperators<Int16,Int16,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<short,short,bool>.operator >= (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IComparisonOperators<Int16,Int16,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<short,short,bool>.operator < (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IComparisonOperators<Int16,Int16,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<short,short,bool>.operator <= (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IDecrementOperators<Int16>.CheckedDecrement(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static short IDecrementOperators<short>.op_CheckedDecrement (short value);
```

Parameters

`value` Int16

The value to decrement.

Returns

Int16

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IDecrementOperators<Int16>.Decrement(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static short IDecrementOperators<short>.operator -- (short value);
```

Parameters

`value` Int16

The value to decrement.

Returns

Int16

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IDivisionOperators<Int16,Int16,Int16>.DivisionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static short IDivisionOperators<short,short,short>.operator / (short left,
short right);
```

Parameters

`left` Int16

The value that `right` divides.

`right` Int16

The value that divides `left`.

Returns

Int16

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IEqualityOperators<Int16,Int16,Boolean>.EqualityOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<short,short,bool>.operator == (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IEqualityOperators<Int16,Int16,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<short,short,bool>.operator != (short left,
short right);
```

Parameters

`left` Int16

The value to compare with `right`.

`right` Int16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IIncrementOperators<Int16>.CheckedIncrement(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static short IIncrementOperators<short>.op_CheckedIncrement (short value);
```

Parameters

`value` Int16

The value to increment.

Returns

Int16

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IIncrementOperators<Int16>.Increment(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static short IIncrementOperators<short>.operator ++ (short value);
```

Parameters

`value` Int16

The value to increment.

Returns

Int16

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IMinMaxValue<Int16>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static short System.Numerics.IMinMaxValue<System.Int16>.MaxValue { get; }
```

Property Value

Int16

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IMinMaxValue<Int16>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static short System.Numerics.IMinMaxValue<System.Int16>.MinValue { get; }
```

Property Value

Int16

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IModulusOperators<Int16,Int16,Int16>.Modulus(Int16, Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static short IModulusOperators<short,short,short>.operator % (short left,
short right);
```

Parameters

left Int16

The value that `right` divides.

right Int16

The value that divides `left`.

Returns

Int16

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IMultiplicativeIdentity<Int16,Int16>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static short
System.Numerics.IMultiplicativeIdentity<System.Int16,System.Int16>.MultiplicativeIdentity { get; }
```

Property Value

Int16

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.IMultiplyOperators<Int16,Int16,Int16>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static short IMultiplyOperators<short,short,short>.op_CheckedMultiply (short
left, short right);
```

Parameters

left Int16

The value that `right` multiplies.

right Int16

The value that multiplies `left`.

Returns

Int16

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IMultiplyOperators<Int16,Int16,Int16>.MultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static short IMultiplyOperators<short,short,short>.operator * (short left,
short right);
```

Parameters

left Int16

The value that `right` multiplies.

right Int16

The value that multiplies `left`.

Returns

Int16

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumber<Int16>.MaxNumber(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static short INumber<short>.MaxNumber (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumber<Int16>.MinNumber(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static short INumber<short>.MinNumber (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsCanonical(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<short>.IsCanonical (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsComplexNumber(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<short>.IsComplexNumber (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsFinite(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<short>.IsFinite (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsImaginaryNumber(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

```
C#
static bool INumberBase<short>.IsImaginaryNumber (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsInfinity(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<short>.IsInfinity (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsInteger(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<short>.IsInteger (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsNaN(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<short>.IsNaN (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsNegativeInfinity(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<short>.IsNegativeInfinity (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsNormal(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<short>.IsNormal (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsPositiveInfinity(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

```
C#
static bool INumberBase<short>.IsPositiveInfinity (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsRealNumber(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<short>.IsRealNumber (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsSubnormal(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<short>.IsSubnormal (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.IsZero(Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<short>.IsZero (short value);
```

Parameters

`value` Int16

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.MaxMagnitudeNumber(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static short INumberBase<short>.MaxMagnitudeNumber (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.MinMagnitudeNumber(Int16, Int16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#
static short INumberBase<short>.MinMagnitudeNumber (short x, short y);
```

Parameters

x Int16

The value to compare with `y`.

y Int16

The value to compare with `x`.

Returns

Int16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static short System.Numerics.INumberBase<System.Int16>.One { get; }
```

Property Value

Int16

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int16.INumberBase<Int16>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Int16>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<short>.TryConvertFromChecked<TOther> (TOther value,
out short result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.INumberBase<Int16>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<short>.TryConvertFromSaturating<TOther> (TOther
value, out short result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.INumberBase<Int16>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<short>.TryConvertFromTruncating<TOther> (TOther
value, out short result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int16.INumberBase<Int16>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<short>.TryConvertToChecked<TOther> (short value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int16.INumberBase<Int16>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<short>.TryConvertToSaturating<TOther> (short value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<short>.TryConvertToTruncating<TOther> (short value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.INumberBase<Int16>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static short System.Numerics.INumberBase<System.Int16>.Zero { get; }
```

Property Value

Int16

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int16.IShift Operators<Int16,Int32,Int16>.Left Shift(Int16, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static short IShiftOperators<short,int,short>.operator << (short value, int
shiftAmount);
```

Parameters

`value` Int16

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Int16

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IShift Operators<Int16,Int32,Int16>.RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static short IShiftOperators<short,int,short>.operator >> (short value, int
shiftAmount);
```

Parameters

`value` Int16

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int16

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IShift Operators<Int16,Int32,Int16>.Unsigned RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static short IShiftOperators<short,int,short>.op_UnsignedRightShift (short
value, int shiftAmount);
```

Parameters

`value` Int16

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int16

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.ISignedNumber<Int16>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

```
C#

static short System.Numerics.ISignedNumber<System.Int16>.NegativeOne { get;
}
```

Property Value

Int16

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int16.ISubtractionOperators<Int16,Int16,Int16>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static short ISubtractionOperators<short,short,short>.op_CheckedSubtraction
(short left, short right);
```

Parameters

`left` Int16

The value from which `right` is subtracted.

`right` Int16

The value that's subtracted from `left`.

Returns

Int16

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.ISubtractionOperators<Int16,Int16,Int16>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static short ISubtractionOperators<short,short,short>.operator - (short
left, short right);
```

Parameters

`left` Int16

The value from which `right` is subtracted.

`right` Int16

The value that's subtracted from `left`.

Returns

Int16

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IUnaryNegationOperators<Int16,Int16>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static short IUnaryNegationOperators<short,short>.op_CheckedUnaryNegation
(short value);
```

Parameters

`value` Int16

The value for which to compute the unary negation.

Returns

Int16

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IUnaryNegationOperators<Int16,Int16>.UnaryNegation(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static short IUnaryNegationOperators<short,short>.operator - (short value);
```

Parameters

`value` Int16

The value for which to compute the unary negation.

Returns

Int16

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int16.IUnaryPlusOperators<Int16,Int16>.UnaryPlus(Int16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static short IUnaryPlusOperators<short,short>.operator + (short value);
```

Parameters

`value` Int16

The value for which to compute the unary plus.

Returns

Int16

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 32-bit signed integer.

C#

```
public readonly struct Int32 : IComparable<int>, IConvertible,
IEquatable<int>, IParsable<int>, ISpanParsable<int>, IUtf8SpanParsable<int>,
System.Numerics.IAdditionOperators<int,int,int>,
System.Numerics.IAdditiveIdentity<int,int>,
System.Numerics.IBinaryInteger<int>, System.Numerics.IBinaryNumber<int>,
System.Numerics.IBitwiseOperators<int,int,int>,
System.Numerics.IComparisonOperators<int,int,bool>,
System.Numerics.IDecrementOperators<int>,
System.Numerics.IDivisionOperators<int,int,int>,
System.Numerics.IEqualityOperators<int,int,bool>,
System.Numerics.IIncrementOperators<int>, System.Numerics.IMinMaxValue<int>,
System.Numerics.IModulusOperators<int,int,int>,
System.Numerics.IMultiplicativeIdentity<int,int>,
System.Numerics.IMultiplyOperators<int,int,int>,
System.Numerics.INumber<int>, System.Numerics.INumberBase<int>,
System.Numerics.IShiftOperators<int,int,int>,
System.Numerics.ISignedNumber<int>,
System.Numerics.ISubtractionOperators<int,int,int>,
System.Numerics.IUnaryNegationOperators<int,int>,
System.Numerics.IUnaryPlusOperators<int,int>
```

Inheritance Object → ValueType → Int32

Implements IComparable , IComparable<Int32> , IConvertible , IEquatable<Int32> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Int32> , IParsable<TSelf> , ISpanParsable<Int32> , ISpanParsable<TSelf> , IAdditionOperators<Int32,Int32,Int32> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Int32,Int32> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Int32> ,

IBinaryNumber<Int32> , IBinaryNumber<TSelf> ,

IBitwiseOperators<Int32,Int32,Int32> ,

IBitwiseOperators<TSelf,TSelf,TSelf> ,

IComparisonOperators<Int32,Int32,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<Int32> , IDecrementOperators<TSelf> ,

IDivisionOperators<Int32,Int32,Int32> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<Int32,Int32,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Int32> ,

IIncrementOperators<TSelf> , IMinMaxValue<Int32> ,

IModulusOperators<Int32,Int32,Int32> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<Int32,Int32> , IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<Int32,Int32,Int32> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Int32> ,

INumber<TSelf> , INumberBase<Int32> , INumberBase<TSelf> ,

IShiftOperators<Int32,Int32,Int32> , IShiftOperators<TSelf,Int32,TSelf> ,

ISignedNumber<Int32> , ISubtractionOperators<Int32,Int32,Int32> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<Int32,Int32> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Int32,Int32> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Int32> ,

IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Int32.

Fields

Expand table

MaxValue	Represents the largest possible value of an Int32. This field is constant.
MinValue	Represents the smallest possible value of Int32. This field is constant.

Methods

Expand table

Abs(Int32)	Computes the absolute of a value.
Clamp(Int32, Int32, Int32)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Int32)	Compares this instance to a specified 32-bit signed integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CopySign(Int32, Int32)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(Int32, Int32)	Computes the quotient and remainder of two values.
Equals(Int32)	Returns a value indicating whether this instance is equal to a specified Int32 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Int32.
IsEvenInteger(Int32)	Determines if a value represents an even integral number.

IsNegative(Int32)	Determines if a value is negative.
IsOddInteger(Int32)	Determines if a value represents an odd integral number.
IsPositive(Int32)	Determines if a value is positive.
IsPow2(Int32)	Determines if a value is a power of two.
LeadingZeroCount(Int32)	Computes the number of leading zeros in a value.
Log2(Int32)	Computes the log2 of a value.
Max(Int32, Int32)	Compares two values to compute which is greater.
MaxMagnitude(Int32, Int32)	Compares two values to compute which is greater.
Min(Int32, Int32)	Compares two values to compute which is lesser.
MinMagnitude(Int32, Int32)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 32-bit signed integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 32-bit signed integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 32-bit signed integer equivalent.
Parse(String, NumberStyles, IFormat Provider)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.
PopCount(Int32)	Computes the number of bits that are set in a value.
RotateLeft(Int32, Int32)	Rotates a value left by a given amount.
RotateRight(Int32, Int32)	Rotates a value right by a given amount.

Sign(Int32)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(Int32)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current integer number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int32)	Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit signed integer equivalent.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int32)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, Int32)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int32)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, Int32)	Tries to parse a string into a value.

TryParse(String, Int32)	Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, NumberStyles, IFormatProvider, Int32)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

Explicit Interface Implementations

Expand table

IAdditionOperators<Int32,Int32,Int32>.Addition(Int32, Int32)	Adds two values together to compute their sum.
IAdditionOperators<Int32,Int32,Int32>.CheckedAddition(Int32, Int32)	Adds two values together to compute their sum.
IAdditiveIdentity<Int32,Int32>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<Int32>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Int32>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Int32>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Int32)	
IBinaryInteger<Int32>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Int32)	
IBinaryInteger<Int32>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Int32>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Int32>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Int32,Int32,Int32>.BitwiseAnd(Int32, Int32)	Computes the bitwise-and of two values.
IBitwiseOperators<Int32,Int32,Int32>.BitwiseOr(Int32, Int32)	Computes the bitwise-or of two values.

IBitwiseOperators<Int32,Int32,Int32>.ExclusiveOr(Int32, Int32)	Computes the exclusive-or of two values.
IBitwiseOperators<Int32,Int32,Int32>.Ones Complement(Int32)	Computes the ones-complement representation of a given value.
IComparisonOperators<Int32,Int32,Boolean>.Greater Than(Int32, Int32)	Compares two values to determine which is greater.
IComparisonOperators<Int32,Int32,Boolean>.Greater ThanOrEqual(Int32, Int32)	Compares two values to determine which is greater or equal.
IComparisonOperators<Int32,Int32,Boolean>.Less Than(Int32, Int32)	Compares two values to determine which is less.
IComparisonOperators<Int32,Int32,Boolean>.LessThan OrEqual(Int32, Int32)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).

IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Int32>.Checked Decrement(Int32)	Decrements a value.
IDecrementOperators<Int32>.Decrement(Int32)	Decrements a value.
IDivisionOperators<Int32,Int32,Int32>.Division(Int32, Int32)	Divides one value by another to compute their quotient.
IEquality Operators<Int32,Int32,Boolean>.Equality(Int32, Int32)	Compares two values to determine equality.
IEquality Operators<Int32,Int32,Boolean>.Inequality(Int32, Int32)	Compares two values to determine inequality.
IIncrementOperators<Int32>.CheckedIncrement(Int32)	Increments a value.
IIncrementOperators<Int32>.Increment(Int32)	Increments a value.
IMinMaxValue<Int32>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Int32>.MinValue	Gets the minimum value of the current type.
IModulusOperators<Int32,Int32,Int32>.Modulus(Int32, Int32)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Int32,Int32>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<Int32,Int32,Int32>.Checked Multiply(Int32, Int32)	Multiplies two values together to compute their product.
IMultiplyOperators<Int32,Int32,Int32>.Multiply(Int32, Int32)	Multiplies two values together to compute their product.
INumber<Int32>.MaxNumber(Int32, Int32)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.

INumber<Int32>.MinNumber(Int32, Int32)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Int32>.IsCanonical(Int32)	Determines if a value is in its canonical representation.
INumberBase<Int32>.IsComplexNumber(Int32)	Determines if a value represents a complex number.
INumberBase<Int32>.IsFinite(Int32)	Determines if a value is finite.
INumberBase<Int32>.IsImaginaryNumber(Int32)	Determines if a value represents a pure imaginary number.
INumberBase<Int32>.IsInfinity(Int32)	Determines if a value is infinite.
INumberBase<Int32>.IsInteger(Int32)	Determines if a value represents an integral number.
INumberBase<Int32>.IsNaN(Int32)	Determines if a value is NaN.
INumberBase<Int32>.IsNegativeInfinity(Int32)	Determines if a value is negative infinity.
INumberBase<Int32>.IsNormal(Int32)	Determines if a value is normal.
INumberBase<Int32>.IsPositiveInfinity(Int32)	Determines if a value is positive infinity.
INumberBase<Int32>.IsRealNumber(Int32)	Determines if a value represents a real number.
INumberBase<Int32>.IsSubnormal(Int32)	Determines if a value is subnormal.
INumberBase<Int32>.IsZero(Int32)	Determines if a value is zero.
INumberBase<Int32>.MaxMagnitudeNumber(Int32, Int32)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Int32>.MinMagnitudeNumber(Int32, Int32)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Int32>.One	Gets the value `1` for the type.
INumberBase<Int32>.Radix	Gets the radix, or base, for the type.
INumberBase<Int32>.TryConvertFromChecked<TOther>(TOther, Int32)	
INumberBase<Int32>.TryConvertFromSaturating<TOther>(TOther, Int32)	

INumberBase<Int32>.TryConvertFromTruncating<TOther>(TOther, Int32)	
INumberBase<Int32>.TryConvertToChecked<TOther>(Int32, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Int32>.TryConvertToSaturating<TOther>(Int32, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Int32>.TryConvertToTruncating<TOther>(Int32, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Int32>.Zero	Gets the value `0` for the type.
IShiftOperators<Int32,Int32,Int32>.LeftShift(Int32, Int32)	Shifts a value left by a given amount.
IShiftOperators<Int32,Int32,Int32>.RightShift(Int32, Int32)	Shifts a value right by a given amount.
IShiftOperators<Int32,Int32,Int32>.UnsignedRightShift(Int32, Int32)	Shifts a value right by a given amount.
ISignedNumber<Int32>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<Int32,Int32,Int32>.CheckedSubtraction(Int32, Int32)	Subtracts two values to compute their difference.
ISubtractionOperators<Int32,Int32,Int32>.Subtraction(Int32, Int32)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Int32,Int32>.CheckedUnaryNegation(Int32)	Computes the checked unary negation of a value.
IUnaryNegationOperators<Int32,Int32>.UnaryNegation(Int32)	Computes the unary negation of a value.
IUnaryPlusOperators<Int32,Int32>.UnaryPlus(Int32)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- UInt32
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of an Int32. This field is constant.

C#

```
public const int MaxValue = 2147483647;
```

Field Value

Value* = *2147483647

Int32

Examples

The following example uses the MaxValue property to prevent an OverflowException when converting to an Int32 value.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      long[] numbersToConvert = { 162345, 32183, -54000, Int64.MaxValue/2 };
      int newNumber;
      foreach (long number in numbersToConvert)
      {
         if (number >= Int32.MinValue && number <= Int32.MaxValue)
         {
            newNumber = Convert.ToInt32(number);
            Console.WriteLine($"Successfully converted {newNumber} to an
Int32.");
         }
         else
         {
```

```
            Console.WriteLine($"Unable to convert {number} to an Int32.");
        }
    }
  }
}
// The example displays the following output to the console:
//    Successfully converted 162345 to an Int32.
//    Successfully converted 32183 to an Int32.
//    Successfully converted -54000 to an Int32.
//    Unable to convert 4611686018427387903 to an Int32.
```

Remarks

The value of this constant is 2,147,483,647; that is, hexadecimal 0x7FFFFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of Int32. This field is constant.

C#

```
public const int MinValue = -2147483648;
```

Field Value

***Value* = -2147483648**

Int32

Examples

The following example uses the MinValue property to prevent an OverflowException when converting to an Int32 value.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      long[] numbersToConvert = { 162345, 32183, -54000, Int64.MaxValue/2 };
      int newNumber;
      foreach (long number in numbersToConvert)
      {
         if (number >= Int32.MinValue && number <= Int32.MaxValue)
         {
            newNumber = Convert.ToInt32(number);
            Console.WriteLine($"Successfully converted {newNumber} to an
Int32.");
         }
         else
         {
```

```
            Console.WriteLine($"Unable to convert {number} to an Int32.");
        }
    }
  }
}
// The example displays the following output to the console:
//    Successfully converted 162345 to an Int32.
//    Successfully converted 32183 to an Int32.
//    Successfully converted -54000 to an Int32.
//    Unable to convert 4611686018427387903 to an Int32.
```

Remarks

The value of this constant is -2,147,483,648; that is, hexadecimal 0x80000000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.Abs(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static int Abs (int value);
```

Parameters

`value` Int32

The value for which to get its absolute.

Returns

Int32

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Clamp(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static int Clamp (int value, int min, int max);
```

Parameters

`value` Int32

The value to clamp.

`min` Int32

The inclusive minimum to which `value` should clamp.

`max` Int32

The inclusive maximum to which `value` should clamp.

Returns

Int32

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified 32-bit signed integer and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Int32)	Compares this instance to a specified 32-bit signed integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(Int32)

Compares this instance to a specified 32-bit signed integer and returns an indication of their relative values.

C#

```
public int CompareTo (int value);
```

Parameters

`value` Int32

An integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following example demonstrates the Int32.CompareTo(Int32) method. In addition to displaying the value returned by the method for four different comparisons, it converts the return value to a member of the custom `Comparison` enumeration, whose value it also displays.

C#

```
using System;

enum Comparison {
   LessThan=-1, Equal=0, GreaterThan=1};

public class ValueComparison
{
   public static void Main()
   {
      int mainValue = 16325;
      int zeroValue = 0;
      int negativeValue = -1934;
      int positiveValue = 903624;
      int sameValue = 16325;

      Console.WriteLine("Comparing {0} and {1}: {2} ({3}).",
                        mainValue, zeroValue,
                        mainValue.CompareTo(zeroValue),
                        (Comparison) mainValue.CompareTo(zeroValue));

      Console.WriteLine("Comparing {0} and {1}: {2} ({3}).",
                        mainValue, sameValue,
                        mainValue.CompareTo(sameValue),
```

```
                        (Comparison) mainValue.CompareTo(sameValue));

        Console.WriteLine("Comparing {0} and {1}: {2} ({3}).",
                          mainValue, negativeValue,
                          mainValue.CompareTo(negativeValue),
                          (Comparison) mainValue.CompareTo(negativeValue));

        Console.WriteLine("Comparing {0} and {1}: {2} ({3}).",
                          mainValue, positiveValue,
                          mainValue.CompareTo(positiveValue),
                          (Comparison) mainValue.CompareTo(positiveValue));
    }
}
// The example displays the following output:
//       Comparing 16325 and 0: 1 (GreaterThan).
//       Comparing 16325 and 16325: 0 (Equal).
//       Comparing 16325 and -1934: 1 (GreaterThan).
//       Comparing 16325 and 903624: -1 (LessThan).
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Int32.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- UInt32
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not an Int32.

Remarks

Any instance of Int32, regardless of its value, is considered greater than `null`.

`value` must be `null` or an instance of Int32; otherwise, an exception is thrown.

See also

- UInt32
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.CopySign(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

```
C#
public static int CopySign (int value, int sign);
```

Parameters

`value` Int32

The value whose magnitude is used in the result.

`sign` Int32

The value whose sign is used in the result.

Returns

Int32

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static int CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int32

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static int CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int32

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.CreateTruncating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static int CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int32

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.DivRem(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#
public static (int Quotient, int Remainder) DivRem (int left, int right);
```

Parameters

`left` Int32

The value which `right` divides.

`right` Int32

The value which divides `left`.

Returns

ValueTuple<Int32,Int32>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified Int32 value.

Overloads

Expand table

Equals(Int32)	Returns a value indicating whether this instance is equal to a specified Int32 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(Int32)

Returns a value indicating whether this instance is equal to a specified Int32 value.

C#

```
public bool Equals (int obj);
```

Parameters

obj Int32

An Int32 value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Int32.Equals(Object) because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int32) method overloads. If an implicit conversion between the `obj` argument and an Int32 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int32) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int32 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, Int16, SByte, and UInt16 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int32) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static int value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,15}", value.Equals(short1));
      TestObjectForEquality(short1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
      TestObjectForEquality(long1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
```

```
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
      Console.WriteLine("value = ushort1: {0,15}",
value.Equals(ushort1));
      TestObjectForEquality(ushort1);

      ulong ulong1 = 112;
      Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
      TestObjectForEquality(ulong1);

      decimal dec1 = 112m;
      Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
      TestObjectForEquality(dec1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }

   private static void TestObjectForEquality(Object obj)
   {
      Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                        value, value.GetType().Name,
                        obj, obj.GetType().Name,
                        value.Equals(obj));
   }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int32) = 112 (Byte): False
//
//       value = short1:            True
//       112 (Int32) = 112 (Int16): False
//
//       value = long1:             False
//       112 (Int32) = 112 (Int64): False
//
//       value = sbyte1:            True
//       112 (Int32) = 112 (SByte): False
//
//       value = ushort1:            True
//       112 (Int32) = 112 (UInt16): False
//
//       value = ulong1:             False
//       112 (Int32) = 112 (UInt64): False
//
//       value = dec1:                False
//       112 (Int32) = 112 (Decimal): False
//
//       value = dbl1:               False
//       112 (Int32) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of Int32 and equals the value of this instance; otherwise, `false`.

Examples

The following example illustrates the use of `Equals` in the context of `Int32`, comparing two `int` values and returning `true` if they represent the same number, or `false` if they do not.

C#

```
Int32 myVariable1 = 60;
Int32 myVariable2 = 60;

// Get and display the declaring type.
Console.WriteLine("\nType of 'myVariable1' is '{0}' and"+
     " value is :{1}",myVariable1.GetType(), myVariable1);
Console.WriteLine("Type of 'myVariable2' is '{0}' and"+
     " value is :{1}",myVariable2.GetType(), myVariable2);

// Compare 'myVariable1' instance with 'myVariable2' Object.
if( myVariable1.Equals( myVariable2 ) )
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are equal");
else
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are not equal");
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int32) method overloads. If an implicit conversion between the `obj` argument and an Int32 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int32) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int32 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, Int16, SByte, and UInt16 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int32) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static int value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,15}", value.Equals(short1));
```

```
        TestObjectForEquality(short1);

        long long1 = 112;
        Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,15}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int32) = 112 (Byte): False
//
//       value = short1:            True
//       112 (Int32) = 112 (Int16): False
//
//       value = long1:             False
//       112 (Int32) = 112 (Int64): False
//
//       value = sbyte1:            True
//       112 (Int32) = 112 (SByte): False
//
//       value = ushort1:            True
//       112 (Int32) = 112 (UInt16): False
//
//       value = ulong1:             False
//       112 (Int32) = 112 (UInt64): False
//
```

```
//       value = dec1:                 False
//       112 (Int32) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (Int32) = 112 (Double): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int32.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Int32.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Int32.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IsEvenInteger(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IsNegative(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IsOddInteger(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (int value);
```

Parameters

value Int32

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IsPositive(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IsPow2(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.LeadingZeroCount(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static int LeadingZeroCount (int value);
```

Parameters

`value` Int32

The value whose leading zeroes are to be counted.

Returns

Int32

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Log2(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static int Log2 (int value);
```

Parameters

`value` Int32

The value whose log2 is to be computed.

Returns

Int32

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Max(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static int Max (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.MaxMagnitude(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static int MaxMagnitude (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Min(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static int Min (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.MinMagnitude(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static int MinMagnitude (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 32-bit signed integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 32-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 32-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 32-bit signed integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.

```C#
public static int Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific information about the format of `s`.

Returns

Int32

A 32-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int32.MinValue or greater than Int32.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example uses a variety of `style` and `provider` parameters to parse the string representations of Int32 values. It also illustrates some of the different ways the same string can be interpreted depending on the culture whose formatting information is used for the parsing operation.

C#

```
using System;
using System.Globalization;

public class ParseInt32
{
   public static void Main()
   {
      Convert("12,000", NumberStyles.Float | NumberStyles.AllowThousands,
              new CultureInfo("en-GB"));
      Convert("12,000", NumberStyles.Float | NumberStyles.AllowThousands,
              new CultureInfo("fr-FR"));
      Convert("12,000", NumberStyles.Float, new CultureInfo("en-US"));

      Convert("12 425,00", NumberStyles.Float |
NumberStyles.AllowThousands,
              new CultureInfo("sv-SE"));
      Convert("12,425.00", NumberStyles.Float |
NumberStyles.AllowThousands,
              NumberFormatInfo.InvariantInfo);
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowDecimalPoint,
              new CultureInfo("fr-FR"));
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowDecimalPoint,
              new CultureInfo("en-US"));
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowThousands,
              new CultureInfo("en-US"));
   }
```

```
    private static void Convert(string value, NumberStyles style,
                                IFormatProvider provider)
    {
       try
       {
          int number = Int32.Parse(value, style, provider);
          Console.WriteLine("Converted '{0}' to {1}.", value, number);
       }
       catch (FormatException)
       {
          Console.WriteLine("Unable to convert '{0}'.", value);
       }
       catch (OverflowException)
       {
          Console.WriteLine("'{0}' is out of range of the Int32 type.",
value);
       }
    }
}
// This example displays the following output to the console:
//       Converted '12,000' to 12000.
//       Converted '12,000' to 12.
//       Unable to convert '12,000'.
//       Converted '12 425,00' to 12425.
//       Converted '12,425.00' to 12425.
//       '631,900' is out of range of the Int32 type.
//       Unable to convert '631,900'.
//       Converted '631,900' to 631900.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digist][e[sign]exponential_digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag or at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific thousands separator symbol. The thousands separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, an OverflowException is thrown.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

Note

Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Int32 type. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements. However, *fractional-digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation. If `s` represents a number in exponential notation, it must represent an integer within the range of the Int32 data type without a non-zero, fractional component.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A positive sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The thousands separator (,) element.
NumberStyles.AllowCurrencySymbol	The *$* element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-

specific information used in parsing. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.

C#

```
public static int Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific information about the format of `s`.

Returns

Int32

A 32-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static int Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int32

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its 32-bit signed integer equivalent.

C#

```
public static int Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Int32

A 32-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not of the correct format.

OverflowException

`s` represents a number less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to an Int32 value.

C#

```
protected void OkToInteger_Click(object sender, EventArgs e)
{
    string locale;
    int number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;
```

```
    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Int32.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

This overload of the Parse(String, IFormatProvider) method is typically used to convert text that can be formatted in a variety of ways to an Int32 value. For example, it can be used to convert the text entered by a user into an HTML text box to a numeric value.

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.

Element	Description
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements that can be present in `s`, use the Int32.Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

Converts the string representation of a number to its 32-bit signed integer equivalent.

```C#
public static int Parse (string s);
```

Parameters

s String

A string containing a number to convert.

Returns

Int32

A 32-bit signed integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Int32.MinValue or greater than Int32.MaxValue.

Examples

The following example demonstrates how to convert a string value into a 32-bit signed integer value using the Int32.Parse(String) method. The resulting integer value is then displayed to the console.

```C#
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "+13230", "-0", "1,390,146",
"$190,235,421,127",
                          "0xFA1B", "163042", "-10", "007", "2147483647",
                          "2147483648", "16e07", "134985.0", "-12034",
```

```
                                "-2147483648", "-2147483649" };
        foreach (string value in values)
        {
           try {
              int number = Int32.Parse(value);
              Console.WriteLine("{0} --> {1}", value, number);
           }
           catch (FormatException) {
              Console.WriteLine("{0}: Bad Format", value);
           }
           catch (OverflowException) {
              Console.WriteLine("{0}: Overflow", value);
           }
        }
     }
  }
  // The example displays the following output:
  //       +13230 --> 13230
  //       -0 --> 0
  //       1,390,146: Bad Format
  //       $190,235,421,127: Bad Format
  //       0xFA1B: Bad Format
  //       163042 --> 163042
  //       -10 --> -10
  //       007 --> 7
  //       2147483647 --> 2147483647
  //       2147483648: Overflow
  //       16e07: Bad Format
  //       134985.0: Bad Format
  //       -12034 --> -12034
  //       -2147483648 --> -2147483648
  //       -2147483649: Overflow
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.

Element	Description
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements that can be present in `s`, use either the Int32.Parse(String, NumberStyles) or the Int32.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see CurrentInfo. To parse a string using the formatting information of some other culture, use the Int32.Parse(String, NumberStyles, IFormatProvider) method.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#
```

```
public static int Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int32

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static int Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int32

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its 32-bit signed integer equivalent.

```
C#

public static int Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

Returns

Int32

A 32-bit signed integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int32.MinValue or greater than Int32.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example uses the Int32.Parse(String, NumberStyles) method to parse the string representations of several Int32 values. The current culture for the example is en-US.

C#

```
using System;
using System.Globalization;

public class ParseInt32
{
```

```
    public static void Main()
    {
       Convert("104.0", NumberStyles.AllowDecimalPoint);
       Convert("104.9", NumberStyles.AllowDecimalPoint);
       Convert(" $17,198,064.42", NumberStyles.AllowCurrencySymbol |
                                  NumberStyles.Number);
       Convert("103E06", NumberStyles.AllowExponent);
       Convert("-1,345,791", NumberStyles.AllowThousands);
       Convert("(1,345,791)", NumberStyles.AllowThousands |
                              NumberStyles.AllowParentheses);
    }

    private static void Convert(string value, NumberStyles style)
    {
       try
       {
          int number = Int32.Parse(value, style);
          Console.WriteLine("Converted '{0}' to {1}.", value, number);
       }
       catch (FormatException)
       {
          Console.WriteLine("Unable to convert '{0}'.", value);
       }
       catch (OverflowException)
       {
          Console.WriteLine("'{0}' is out of range of the Int32 type.",
value);
       }
    }
}
// The example displays the following output to the console:
//       Converted '104.0' to 104.
//       '104.9' is out of range of the Int32 type.
//       ' $17,198,064.42' is out of range of the Int32 type.
//       Converted '103E06' to 103000000.
//       Unable to convert '-1,345,791'.
//       Converted '(1,345,791)' to -1345791.
```

Remarks

The `style` parameter defines the style elements (such as white space, the positive or negative sign symbol, or the thousands separator symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]exponential_digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific thousands separator symbol. The current culture's thousands separator can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, an OverflowException is thrown.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfull if it is in the range of the Int32 type. Most of the remaining NumberStyles members control elements that may be but are not required to be present in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (,), and decimal point (.) elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be combined with the `s` parameter it are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Int32.Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int32.PopCount(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static int PopCount (int value);
```

Parameters

`value` Int32

The value whose set bits are to be counted.

Returns

Int32

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.RotateLeft(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static int RotateLeft (int value, int rotateAmount);
```

Parameters

`value` Int32

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Int32

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.RotateRight(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
public static int RotateRight (int value, int rotateAmount);
```

Parameters

`value` Int32

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Int32

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.Sign(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (int value);
```

Parameters

`value` Int32

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a negative sign if the value is negative, and a sequence of digits ranging from 0 to 9 with no leading zeroes.

Examples

The following example displays an Int32 value using the default ToString() method. It also displays the string representations of the Int32 value that results from using a number of standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
int value = -16325;
// Display value using default ToString method.
Console.WriteLine(value.ToString());            // Displays -16325
// Display value using some standard format specifiers.
Console.WriteLine(value.ToString("G"));         // Displays -16325
Console.WriteLine(value.ToString("C"));         // Displays ($16,325.00)
Console.WriteLine(value.ToString("D"));         // Displays -16325
Console.WriteLine(value.ToString("F"));         // Displays -16325.00
Console.WriteLine(value.ToString("N"));         // Displays -16,325.00
Console.WriteLine(value.ToString("X"));         // Displays FFFFC03B
```

Remarks

The ToString() method formats an Int32 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example displays the string representation of an Int32 value using CultureInfo objects that represent several different cultures.

C#

```
int value = -16325;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));
// This example displays the following output to the console:
//       -16325
//       -16325
//       -16325
```

Remarks

The ToString(IFormatProvider) method formats an Int32 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider and whose GetFormat method returns a NumberFormatInfo object that provides formatting information.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value is formatted using the NumberFormatInfo object for the thread current culture. For information about the thread current culture, see Thread.CurrentCulture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid or not supported.

Examples

The following example displays an Int32 value using each of the supported standard numeric format specifiers, together with two custom numeric format strings. In converting the numeric values to strings, the example uses the formatting conventions of the en-US culture.

C#

```
int value = -16325;
string specifier;

// Use standard numeric format specifier.
specifier = "G";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    G: -16325
specifier = "C";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    C: ($16,325.00)
specifier = "D8";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    D8: -00016325
specifier = "E4";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    E4: -1.6325E+004
specifier = "e3";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    e3: -1.633e+004
specifier = "F";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    F: -16325.00
specifier = "N";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    N: -16,325.00
specifier = "P";
Console.WriteLine("{0}: {1}", specifier,
(value/100000).ToString(specifier));
// Displays:    P: -16.33 %
specifier = "X";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    X: FFFFC03B

// Use custom numeric format specifiers.
specifier = "0,0.000";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    0,0.000: -16,325.000
specifier = "#,#.00#;(#,#.00#)";
Console.WriteLine("{0}: {1}", specifier, (value*-1).ToString(specifier));
// Displays:    #,#.00#;(#,#.00#): 16,325.00
```

Remarks

The ToString(String) method formats an Int32 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid standard numeric format specifier except for "R", as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string (""), the return value of this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The return value of this instance is formatted with the NumberFormatInfo for the current culture.

See also

- Parse(String)
- String
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A standard or custom numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid or not supported.

Examples

The following example displays a positive and a negative value using each of the supported standard numeric format specifiers for three different cultures.

C#

```
// Define cultures whose formatting conventions are to be used.
CultureInfo[] cultures = {CultureInfo.CreateSpecificCulture("en-US"),
                          CultureInfo.CreateSpecificCulture("fr-FR"),
                          CultureInfo.CreateSpecificCulture("es-ES") };
int positiveNumber = 1679;
int negativeNumber = -3045;
string[] specifiers = {"G", "C", "D8", "E2", "F", "N", "P", "X8"};

foreach (string specifier in specifiers)
{
   foreach (CultureInfo culture in cultures)
   {
      // Display values with "G" format specifier.
      Console.WriteLine("{0} format using {1} culture: {2, 16} {3, 16}",
                        specifier, culture.Name,
                        positiveNumber.ToString(specifier, culture),
                        negativeNumber.ToString(specifier, culture));
   }
   Console.WriteLine();
}
// The example displays the following output:
//       G format using en-US culture:             1679            -3045
//       G format using fr-FR culture:             1679            -3045
//       G format using es-ES culture:             1679            -3045
//
//       C format using en-US culture:        $1,679.00      ($3,045.00)
//       C format using fr-FR culture:       1 679,00 €      -3 045,00 €
//       C format using es-ES culture:       1.679,00 €      -3.045,00 €
//
//       D8 format using en-US culture:         00001679        -00003045
//       D8 format using fr-FR culture:         00001679        -00003045
//       D8 format using es-ES culture:         00001679        -00003045
//
//       E2 format using en-US culture:        1.68E+003       -3.05E+003
//       E2 format using fr-FR culture:        1,68E+003       -3,05E+003
//       E2 format using es-ES culture:        1,68E+003       -3,05E+003
//
//       F format using en-US culture:          1679.00         -3045.00
//       F format using fr-FR culture:          1679,00         -3045,00
//       F format using es-ES culture:          1679,00         -3045,00
//
//       N format using en-US culture:         1,679.00        -3,045.00
//       N format using fr-FR culture:         1 679,00        -3 045,00
//       N format using es-ES culture:         1.679,00        -3.045,00
//
//       P format using en-US culture:     167,900.00 %    -304,500.00 %
//       P format using fr-FR culture:     167 900,00 %    -304 500,00 %
//       P format using es-ES culture:     167.900,00 %    -304.500,00 %
//
```

```
//       X8 format using en-US culture:         0000068F         FFFFF41B
//       X8 format using fr-FR culture:         0000068F         FFFFF41B
//       X8 format using es-ES culture:         0000068F         FFFFF41B
```

Remarks

The ToString(String, IFormatProvider) method formats an Int32 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string (""), the return value for this instance is formatted with the general numeric format specifier ("G").

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific format information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider and whose GetFormat method returns a NumberFormatInfo object that provides formatting information.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value for this instance is formatted with the NumberFormatInfo

for the current culture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Int32.TrailingZeroCount(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static int TrailingZeroCount (int value);
```

Parameters

`value` Int32

The value whose trailing zeroes are to be counted.

Returns

Int32

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current integer number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current integer number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the operation succeeded.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Int32)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, Int32)	Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int32)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Int32)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int32)	Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit signed integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int32)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

TryParse(String, NumberStyles, IFormatProvider, Int32)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int32)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out int result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int32

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Int32)

Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out int result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert.

`result` Int32

When this method returns, contains the 32-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int32.MinValue or greater than Int32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Int32)

Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

```C#
public static bool TryParse (string? s, out int result);
```

Parameters

`s` String

A string containing a number to convert.

`result` Int32

When this method returns, contains the 32-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Int32.MinValue or greater than Int32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the Int32.TryParse(String, Int32) method with a number of different string values.

```C#
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { null, "160519", "9432.0", "16,667",
                          "   -322   ", "+4302", "(100);", "01FA" };
      foreach (var value in values)
      {
         int number;
```

```
            bool success = int.TryParse(value, out number);
            if (success)
            {
               Console.WriteLine($"Converted '{value}' to {number}.");
            }
            else
            {
               Console.WriteLine($"Attempted conversion of '{value ?? "
<null>"}' failed.");
            }
         }
      }
   }
// The example displays the following output:
//       Attempted conversion of '<null>' failed.
//       Converted '160519' to 160519.
//       Attempted conversion of '9432.0' failed.
//       Attempted conversion of '16,667' failed.
//       Converted '   -322   ' to -322.
//       Converted '+4302' to 4302.
//       Attempted conversion of '(100);' failed.
//       Attempted conversion of '01FA' failed.
```

Some of the strings that the TryParse(String, Int32) method is unable to convert in this example are:

- "9432.0". The conversion fails because the string cannot contain a decimal separator; it must contain integral digits only.
- "16,667". The conversion fails because the string cannot contain group separators; it must contain integral digits only.
- "(100)". The conversion fails because the string cannot contain a negative sign other than the one defined by the current culture's NumberFormatInfo.NegativeSign and NumberFormatInfo.NumberNegativePattern properties.
- "01FA". The conversion fails because the string cannot contain hexadecimal digits; it must contain decimal digits only.

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Int32.TryParse(String, NumberStyles, IFormatProvider, Int32) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see CurrentInfo.

This overload of the TryParse method interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the Int32.TryParse(String, NumberStyles, IFormatProvider, Int32) overload.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, Int32)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out int result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int32

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Int32)

Tries to parse a string into a value.

```
C#
public static bool TryParse (string? s, IFormatProvider? provider, out
int result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int32

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int32)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
int result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int32

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Int32)

Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit signed integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out int
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Int32

When this method returns, contains the 32-bit signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int32)

Converts the span representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates

whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
int result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Int32

When this method returns, contains the 32-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int32.MinValue or greater than Int32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Int32)

Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out int result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Int32

When this method returns, contains the 32-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int32.MinValue or greater than Int32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the Int32.TryParse(String, NumberStyles, IFormatProvider, Int32) method with a number of different string and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class StringParsing
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "106779";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-30677";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);

      styles = NumberStyles.AllowLeadingSign;
      CallTryParse(numericString, styles);

      numericString = "301677-";
      CallTryParse(numericString, styles);

      styles = styles | NumberStyles.AllowTrailingSign;
      CallTryParse(numericString, styles);

      numericString = "$10634";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      styles = NumberStyles.Integer | NumberStyles.AllowCurrencySymbol;
```

```
      CallTryParse(numericString, styles);

      numericString = "10345.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "22,593";
      styles = NumberStyles.Integer | NumberStyles.AllowThousands;
      CallTryParse(numericString, styles);

      numericString = "12E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "12E03";
      CallTryParse(numericString, styles);

      numericString = "80c1";
      CallTryParse(numericString, NumberStyles.HexNumber);

      numericString = "0x80C1";
      CallTryParse(numericString, NumberStyles.HexNumber);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      CultureInfo provider;

      // If currency symbol is allowed, use en-US culture.
      if ((styles & NumberStyles.AllowCurrencySymbol) > 0)
         provider = new CultureInfo("en-US");
      else
         provider = CultureInfo.InvariantCulture;

      bool success = int.TryParse(stringToConvert, styles,
                                  provider, out int number);
      if (success)
         Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
      else
         Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
   }
}
// The example displays the following output to the console:
//       Converted '106779' to 106779.
//       Attempted conversion of '-30677' failed.
//       Converted '-30677' to -30677.
//       Attempted conversion of '301677-' failed.
//       Converted '301677-' to -301677.
```

```
//       Attempted conversion of '$10634' failed.
//       Converted '$10634' to 10634.
//       Converted '10345.00' to 10345.
//       Attempted conversion of '10345.72' failed.
//       Converted '22,593' to 22593.
//       Attempted conversion of '12E-01' failed.
//       Converted '12E03' to 12000.
//       Converted '80c1' to 32961.
//       Attempted conversion of '0x80C1' failed.
```

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]digits][ws]

Or, if the `style` parameter includes AllowHexSpecifier:

[ws]hexdigits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.

Element	Description
sign	An optional sign. A sign symbol can appear in `s` if `style` includes the NumberStyles.AllowLeadingSign or NumberStyles.AllowTrailingSign flags.
digits	A sequence of digits from 0 through 9.
,	A culture-specific thousands separator. The thousands separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or the method returns `false`.

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation. If `s` represents a number in exponential notation, it must represent an integer within the range of the Int32 data type without a non-zero, fractional component.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The thousands separator (,) element.
NumberStyles.AllowCurrencySymbol	The *$* element.
NumberStyles.Currency	All elements. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
NumberStyles.Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
NumberStyles.Number	The *ws*, *sign*, thousands separator (,), and decimal point (.) elements.
NumberStyles.Any	All styles, except `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a CultureInfo object or a NumberFormatInfo object, whose GetFormat method returns a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific

information about the format of `s`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String)
- NumberStyles
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

```
C#
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

```
C#
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.To Int32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, unchanged.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.To Int64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToSByte(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

```
C#
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The type to which to convert this Int32 value.

`provider` IFormatProvider

An object that provides information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method instead.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt16(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt32(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int32 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt64(Int32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.IAdditionOperators<Int32,Int32,Int32>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#
static int IAdditionOperators<int,int,int>.operator + (int left, int right);
```

Parameters

`left` Int32

The value to which `right` is added.

`right` Int32

The value that's added to `left`.

Returns

Int32

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IAdditionOperators<Int32,Int32,Int32>.CheckedAdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static int IAdditionOperators<int,int,int>.op_CheckedAddition (int left, int right);
```

Parameters

`left` Int32

The value to which `right` is added.

`right` Int32

The value that's added to `left`.

Returns

Int32

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IAdditiveIdentity<Int32,Int32>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static int
System.Numerics.IAdditiveIdentity<System.Int32,System.Int32>.AdditiveIdentity { get; }
```

Property Value

Int32

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.IBinaryInteger<Int32>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<int>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.IBinaryInteger<Int32>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<int>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int32.IBinaryInteger<Int32>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<int>.TryReadBigEndian (ReadOnlySpan<byte> source,
bool isUnsigned, out int value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int32.IBinaryInteger<Int32>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<int>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out int value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int32.IBinaryInteger<Int32>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<int>.TryWriteBigEndian (Span<byte> destination, out int
bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBinaryInteger<Int32>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<int>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBinaryNumber<Int32>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static int System.Numerics.IBinaryNumber<System.Int32>.AllBitsSet { get; }
```

Property Value

Int32

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBitwiseOperators<Int32,Int32,Int32>.Bitwise And Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#
static int IBitwiseOperators<int,int,int>.operator & (int left, int right);
```

Parameters

`left` Int32

The value to `and` with `right`.

`right` Int32

The value to `and` with `left`.

Returns

Int32

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBitwiseOperators<Int32,Int32,Int32>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#
static int IBitwiseOperators<int,int,int>.operator | (int left, int right);
```

Parameters

`left` Int32

The value to `or` with `right`.

`right` Int32

The value to `or` with `left`.

Returns

Int32

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBitwise Operators<Int32,Int32,Int32>.Exclusive Or Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#
static int IBitwiseOperators<int,int,int>.operator ^ (int left, int right);
```

Parameters

`left` Int32

The value to xor with `right`.

`right` Int32

The value to xor with `left`.

Returns

Int32

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IBitwiseOperators<Int32,Int32,Int32>.OnesComplement(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static int IBitwiseOperators<int,int,int>.operator ~ (int value);
```

Parameters

`value` Int32

The value for which to compute the ones-complement.

Returns

Int32

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IComparisonOperators<Int32,Int32,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<int,int,bool>.operator > (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IComparisonOperators<Int32,Int32,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<int,int,bool>.operator >= (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IComparisonOperators<Int32,Int32,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<int,int,bool>.operator < (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IComparisonOperators<Int32,Int32,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<int,int,bool>.operator <= (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IDecrementOperators<Int32>.CheckedDecrement(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static int IDecrementOperators<int>.op_CheckedDecrement (int value);
```

Parameters

`value` Int32

The value to decrement.

Returns

Int32

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IDecrementOperators<Int32>.Decrement(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static int IDecrementOperators<int>.operator -- (int value);
```

Parameters

`value` Int32

The value to decrement.

Returns

Int32

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IDivisionOperators<Int32,Int32,Int32>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#
static int IDivisionOperators<int,int,int>.operator / (int left, int right);
```

Parameters

`left` Int32

The value that `right` divides.

`right` Int32

The value that divides `left`.

Returns

Int32

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IEqualityOperators<Int32,Int32,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<int,int,bool>.operator == (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IEqualityOperators<Int32,Int32,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<int,int,bool>.operator != (int left, int right);
```

Parameters

`left` Int32

The value to compare with `right`.

`right` Int32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IIncrementOperators<Int32>.CheckedIncrement(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static int IIncrementOperators<int>.op_CheckedIncrement (int value);
```

Parameters

`value` Int32

The value to increment.

Returns

Int32

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IIncrementOperators<Int32>.Increment(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static int IIncrementOperators<int>.operator ++ (int value);
```

Parameters

`value` Int32

The value to increment.

Returns

Int32

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IMinMaxValue<Int32>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static int System.Numerics.IMinMaxValue<System.Int32>.MaxValue { get; }
```

Property Value

Int32

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IMinMaxValue<Int32>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static int System.Numerics.IMinMaxValue<System.Int32>.MinValue { get; }
```

Property Value

Int32

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IModulusOperators<Int32,Int32,Int32>.Modulus(Int32, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#
static int IModulusOperators<int,int,int>.operator % (int left, int right);
```

Parameters

`left` Int32

The value that `right` divides.

`right` Int32

The value that divides `left`.

Returns

Int32

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IMultiplicativeIdentity<Int32,Int32>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static int
System.Numerics.IMultiplicativeIdentity<System.Int32,System.Int32>.MultiplicativeIdentity { get; }
```

Property Value

Int32

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.IMultiplyOperators<Int32,Int32,Int32>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static int IMultiplyOperators<int,int,int>.op_CheckedMultiply (int left, int
right);
```

Parameters

left Int32

The value that `right` multiplies.

right Int32

The value that multiplies `left`.

Returns

Int32

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IMultiplyOperators<Int32,Int32,Int32>.MultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#
static int IMultiplyOperators<int,int,int>.operator * (int left, int right);
```

Parameters

`left` Int32

The value that `right` multiplies.

`right` Int32

The value that multiplies `left`.

Returns

Int32

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumber<Int32>.MaxNumber(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static int INumber<int>.MaxNumber (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumber<Int32>.MinNumber(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static int INumber<int>.MinNumber (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsCanonical(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<int>.IsCanonical (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsComplexNumber(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<int>.IsComplexNumber (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsFinite(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<int>.IsFinite (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsImaginaryNumber(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

```
C#
static bool INumberBase<int>.IsImaginaryNumber (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsInfinity(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<int>.IsInfinity (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsInteger(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<int>.IsInteger (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsNaN(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<int>.IsNaN (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsNegativeInfinity(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<int>.IsNegativeInfinity (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsNormal(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<int>.IsNormal (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsPositiveInfinity(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

```
C#
static bool INumberBase<int>.IsPositiveInfinity (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsRealNumber(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<int>.IsRealNumber (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsSubnormal(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<int>.IsSubnormal (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.IsZero(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<int>.IsZero (int value);
```

Parameters

`value` Int32

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.MaxMagnitudeNumber(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static int INumberBase<int>.MaxMagnitudeNumber (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.MinMagnitudeNumber(Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static int INumberBase<int>.MinMagnitudeNumber (int x, int y);
```

Parameters

x Int32

The value to compare with `y`.

y Int32

The value to compare with `x`.

Returns

Int32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static int System.Numerics.INumberBase<System.Int32>.One { get; }
```

Property Value

Int32

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int32.INumberBase<Int32>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Int32>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<int>.TryConvertFromChecked<TOther> (TOther value,
out int result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.INumberBase<Int32>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<int>.TryConvertFromSaturating<TOther> (TOther value,
out int result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.INumberBase<Int32>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<int>.TryConvertFromTruncating<TOther> (TOther value,
out int result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int32.INumberBase<Int32>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<int>.TryConvertToChecked<TOther> (int value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int32.INumberBase<Int32>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<int>.TryConvertToSaturating<TOther> (int value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<int>.TryConvertToTruncating<TOther> (int value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.INumberBase<Int32>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
static int System.Numerics.INumberBase<System.Int32>.Zero { get; }
```

Property Value

Int32

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int32.IShift Operators<Int32,Int32,Int32>.Left Shift(Int32, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static int IShiftOperators<int,int,int>.operator << (int value, int
shiftAmount);
```

Parameters

`value` Int32

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Int32

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IShift Operators<Int32,Int32,Int32>.RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static int IShiftOperators<int,int,int>.operator >> (int value, int
shiftAmount);
```

Parameters

`value` Int32

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int32

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IShift Operators<Int32,Int32,Int32>.Unsigned RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static int IShiftOperators<int,int,int>.op_UnsignedRightShift (int value,
int shiftAmount);
```

Parameters

`value` Int32

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int32

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.ISignedNumber<Int32>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
static int System.Numerics.ISignedNumber<System.Int32>.NegativeOne { get; }
```

Property Value

Int32

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int32.ISubtractionOperators<Int32,Int32,Int32>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static int ISubtractionOperators<int,int,int>.op_CheckedSubtraction (int
left, int right);
```

Parameters

`left` Int32

The value from which `right` is subtracted.

`right` Int32

The value that's subtracted from `left`.

Returns

Int32

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.ISubtractionOperators<Int32,Int32,Int32>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static int ISubtractionOperators<int,int,int>.operator - (int left, int
right);
```

Parameters

`left` Int32

The value from which `right` is subtracted.

`right` Int32

The value that's subtracted from `left`.

Returns

Int32

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IUnaryNegationOperators<Int32,Int32>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static int IUnaryNegationOperators<int,int>.op_CheckedUnaryNegation (int
value);
```

Parameters

`value` Int32

The value for which to compute the unary negation.

Returns

Int32

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IUnaryNegationOperators<Int32,Int32>.UnaryNegation(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static int IUnaryNegationOperators<int,int>.operator - (int value);
```

Parameters

`value` Int32

The value for which to compute the unary negation.

Returns

Int32

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int32.IUnaryPlusOperators<Int32,Int32>.UnaryPlus(Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static int IUnaryPlusOperators<int,int>.operator + (int value);
```

Parameters

`value` Int32

The value for which to compute the unary plus.

Returns

Int32

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 64-bit signed integer.

C#

```
public readonly struct Int64 : IComparable<long>, IConvertible,
IEquatable<long>, IParsable<long>, ISpanParsable<long>,
IUtf8SpanParsable<long>, System.Numerics.IAdditionOperators<long,long,long>,
System.Numerics.IAdditiveIdentity<long,long>,
System.Numerics.IBinaryInteger<long>, System.Numerics.IBinaryNumber<long>,
System.Numerics.IBitwiseOperators<long,long,long>,
System.Numerics.IComparisonOperators<long,long,bool>,
System.Numerics.IDecrementOperators<long>,
System.Numerics.IDivisionOperators<long,long,long>,
System.Numerics.IEqualityOperators<long,long,bool>,
System.Numerics.IIncrementOperators<long>,
System.Numerics.IMinMaxValue<long>,
System.Numerics.IModulusOperators<long,long,long>,
System.Numerics.IMultiplicativeIdentity<long,long>,
System.Numerics.IMultiplyOperators<long,long,long>,
System.Numerics.INumber<long>, System.Numerics.INumberBase<long>,
System.Numerics.IShiftOperators<long,int,long>,
System.Numerics.ISignedNumber<long>,
System.Numerics.ISubtractionOperators<long,long,long>,
System.Numerics.IUnaryNegationOperators<long,long>,
System.Numerics.IUnaryPlusOperators<long,long>
```

Inheritance Object → ValueType → Int64

Implements IComparable , IComparable<Int64> , IConvertible , IEquatable<Int64> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Int64> , IParsable<TSelf> , ISpanParsable<Int64> , ISpanParsable<TSelf> , IAdditionOperators<Int64,Int64,Int64> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Int64,Int64> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<Int64> ,

IBinaryNumber<Int64> , IBinaryNumber<TSelf> ,

IBitwiseOperators<Int64,Int64,Int64> ,

IBitwiseOperators<TSelf,TSelf,TSelf> ,

IComparisonOperators<Int64,Int64,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<Int64> , IDecrementOperators<TSelf> ,

IDivisionOperators<Int64,Int64,Int64> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<Int64,Int64,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<Int64> ,

IIncrementOperators<TSelf> , IMinMaxValue<Int64> ,

IModulusOperators<Int64,Int64,Int64> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<Int64,Int64> , IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<Int64,Int64,Int64> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Int64> ,

INumber<TSelf> , INumberBase<Int64> , INumberBase<TSelf> ,

IShiftOperators<Int64,Int32,Int64> , IShiftOperators<TSelf,Int32,TSelf> ,

ISignedNumber<Int64> , ISubtractionOperators<Int64,Int64,Int64> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<Int64,Int64> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Int64,Int64> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Int64> ,

IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Int64.

Fields

Expand table

MaxValue	Represents the largest possible value of an Int64. This field is constant.
MinValue	Represents the smallest possible value of an Int64. This field is constant.

Methods

Expand table

Abs(Int64)	Computes the absolute of a value.
Clamp(Int64, Int64, Int64)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Int64)	Compares this instance to a specified 64-bit signed integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CopySign(Int64, Int64)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(Int64, Int64)	Computes the quotient and remainder of two values.
Equals(Int64)	Returns a value indicating whether this instance is equal to a specified Int64 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Int64.
IsEvenInteger(Int64)	Determines if a value represents an even integral number.

IsNegative(Int64)	Determines if a value is negative.
IsOddInteger(Int64)	Determines if a value represents an odd integral number.
IsPositive(Int64)	Determines if a value is positive.
IsPow2(Int64)	Determines if a value is a power of two.
LeadingZeroCount(Int64)	Computes the number of leading zeros in a value.
Log2(Int64)	Computes the log2 of a value.
Max(Int64, Int64)	Compares two values to compute which is greater.
MaxMagnitude(Int64, Int64)	Compares two values to compute which is greater.
Min(Int64, Int64)	Compares two values to compute which is lesser.
MinMagnitude(Int64, Int64)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 64-bit signed integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 64-bit signed integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 64-bit signed integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.
PopCount(Int64)	Computes the number of bits that are set in a value.
RotateLeft(Int64, Int32)	Rotates a value left by a given amount.
RotateRight(Int64, Int32)	Rotates a value right by a given amount.

Sign(Int64)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(Int64)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current long number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int64)	Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit signed integer equivalent.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int64)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, Int64)	Converts the span representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int64)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Int64)	Tries to parse a string into a value.

TryParse(String, Int64)	Converts the string representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Int64)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<Int64,Int64,Int64>.Addition(Int64, Int64)	Adds two values together to compute their sum.
IAdditionOperators<Int64,Int64,Int64>.Checked Addition(Int64, Int64)	Adds two values together to compute their sum.
IAdditiveIdentity<Int64,Int64>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<Int64>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Int64>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Int64>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Int64)	
IBinaryInteger<Int64>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Int64)	
IBinaryInteger<Int64>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Int64>.TryWriteLittle Endian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Int64>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Int64,Int64,Int64>.BitwiseAnd(Int64, Int64)	Computes the bitwise-and of two values.
IBitwiseOperators<Int64,Int64,Int64>.BitwiseOr(Int64, Int64)	Computes the bitwise-or of two values.

IBitwiseOperators<Int64,Int64,Int64>.ExclusiveOr(Int64, Int64)	Computes the exclusive-or of two values.
IBitwiseOperators<Int64,Int64,Int64>.Ones Complement(Int64)	Computes the ones-complement representation of a given value.
IComparisonOperators<Int64,Int64,Boolean>.Greater Than(Int64, Int64)	Compares two values to determine which is greater.
IComparisonOperators<Int64,Int64,Boolean>.Greater ThanOrEqual(Int64, Int64)	Compares two values to determine which is greater or equal.
IComparisonOperators<Int64,Int64,Boolean>.Less Than(Int64, Int64)	Compares two values to determine which is less.
IComparisonOperators<Int64,Int64,Boolean>.LessThan OrEqual(Int64, Int64)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).

IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Int64>.Checked Decrement(Int64)	Decrements a value.
IDecrementOperators<Int64>.Decrement(Int64)	Decrements a value.
IDivisionOperators<Int64,Int64,Int64>.Division(Int64, Int64)	Divides one value by another to compute their quotient.
IEquality Operators<Int64,Int64,Boolean>.Equality(Int64, Int64)	Compares two values to determine equality.
IEquality Operators<Int64,Int64,Boolean>.Inequality(Int64, Int64)	Compares two values to determine inequality.
IIncrementOperators<Int64>.CheckedIncrement(Int64)	Increments a value.
IIncrementOperators<Int64>.Increment(Int64)	Increments a value.
IMinMaxValue<Int64>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Int64>.MinValue	Gets the minimum value of the current type.
IModulusOperators<Int64,Int64,Int64>.Modulus(Int64, Int64)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Int64,Int64>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<Int64,Int64,Int64>.Checked Multiply(Int64, Int64)	Multiplies two values together to compute their product.
IMultiplyOperators<Int64,Int64,Int64>.Multiply(Int64, Int64)	Multiplies two values together to compute their product.
INumber<Int64>.MaxNumber(Int64, Int64)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.

INumber<Int64>.MinNumber(Int64, Int64)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<Int64>.IsCanonical(Int64)	Determines if a value is in its canonical representation.
INumberBase<Int64>.IsComplexNumber(Int64)	Determines if a value represents a complex number.
INumberBase<Int64>.IsFinite(Int64)	Determines if a value is finite.
INumberBase<Int64>.IsImaginaryNumber(Int64)	Determines if a value represents a pure imaginary number.
INumberBase<Int64>.IsInfinity(Int64)	Determines if a value is infinite.
INumberBase<Int64>.IsInteger(Int64)	Determines if a value represents an integral number.
INumberBase<Int64>.IsNaN(Int64)	Determines if a value is NaN.
INumberBase<Int64>.IsNegativeInfinity(Int64)	Determines if a value is negative infinity.
INumberBase<Int64>.IsNormal(Int64)	Determines if a value is normal.
INumberBase<Int64>.IsPositiveInfinity(Int64)	Determines if a value is positive infinity.
INumberBase<Int64>.IsRealNumber(Int64)	Determines if a value represents a real number.
INumberBase<Int64>.IsSubnormal(Int64)	Determines if a value is subnormal.
INumberBase<Int64>.IsZero(Int64)	Determines if a value is zero.
INumberBase<Int64>.MaxMagnitudeNumber(Int64, Int64)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<Int64>.MinMagnitudeNumber(Int64, Int64)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<Int64>.One	Gets the value `1` for the type.
INumberBase<Int64>.Radix	Gets the radix, or base, for the type.
INumberBase<Int64>.TryConvertFromChecked<TOther>(TOther, Int64)	
INumberBase<Int64>.TryConvertFromSaturating<TOther>(TOther, Int64)	

INumberBase<Int64>.TryConvertFromTruncating<TOther>(TOther, Int64)	
INumberBase<Int64>.TryConvertToChecked<TOther>(Int64, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Int64>.TryConvertToSaturating<TOther>(Int64, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<Int64>.TryConvertToTruncating<TOther>(Int64, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Int64>.Zero	Gets the value `0` for the type.
IShiftOperators<Int64,Int32,Int64>.LeftShift(Int64, Int32)	Shifts a value left by a given amount.
IShiftOperators<Int64,Int32,Int64>.RightShift(Int64, Int32)	Shifts a value right by a given amount.
IShiftOperators<Int64,Int32,Int64>.UnsignedRightShift(Int64, Int32)	Shifts a value right by a given amount.
ISignedNumber<Int64>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<Int64,Int64,Int64>.CheckedSubtraction(Int64, Int64)	Subtracts two values to compute their difference.
ISubtractionOperators<Int64,Int64,Int64>.Subtraction(Int64, Int64)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Int64,Int64>.CheckedUnaryNegation(Int64)	Computes the checked unary negation of a value.
IUnaryNegationOperators<Int64,Int64>.UnaryNegation(Int64)	Computes the unary negation of a value.
IUnaryPlusOperators<Int64,Int64>.UnaryPlus(Int64)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- UInt64
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of an Int64. This field is constant.

```
C#
public const long MaxValue = 9223372036854775807;
```

Field Value

Value = 9223372036854775807

Int64

Remarks

The value of this constant is 9,223,372,036,854,775,807; that is, hexadecimal 0x7FFFFFFFFFFFFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of an Int64. This field is constant.

C#

```
public const long MinValue = -9223372036854775808;
```

Field Value

***Value* = -9223372036854775808**

Int64

Remarks

The value of this constant is negative 9,223,372,036,854,775,808; that is, hexadecimal 0x8000000000000000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Abs(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static long Abs (long value);
```

Parameters

`value` Int64

The value for which to get its absolute.

Returns

Int64

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Clamp(Int64, Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static long Clamp (long value, long min, long max);
```

Parameters

`value` Int64

The value to clamp.

`min` Int64

The inclusive minimum to which `value` should clamp.

`max` Int64

The inclusive maximum to which `value` should clamp.

Returns

Int64

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or Int64 and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Int64)	Compares this instance to a specified 64-bit signed integer and returns an indication of their relative values.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(Int64)

Compares this instance to a specified 64-bit signed integer and returns an indication of their relative values.

C#

```
public int CompareTo (long value);
```

Parameters

`value` Int64

An integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");
```

```
    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
```

```
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                            resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
```

```
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the Int64.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- UInt64

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.

Return Value	Description
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not an Int64.

Remarks

An Int64, regardless of its value, is considered greater than a null reference.

The `value` parameter must be `null` or an instance of Int64; otherwise, an exception is thrown.

See also

- UInt64
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.CopySign(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

```
C#
public static long CopySign (long value, long sign);
```

Parameters

`value` Int64

The value whose magnitude is used in the result.

`sign` Int64

The value whose sign is used in the result.

Returns

Int64

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static long CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int64

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static long CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int64

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.CreateTruncating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static long CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Int64

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.DivRem(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (long Quotient, long Remainder) DivRem (long left, long
right);
```

Parameters

`left` Int64

The value which `right` divides.

`right` Int64

The value which divides `left`.

Returns

ValueTuple<Int64,Int64>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or Int64.

Overloads

Expand table

Equals(Int64)	Returns a value indicating whether this instance is equal to a specified Int64 value.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(Int64)

Returns a value indicating whether this instance is equal to a specified Int64 value.

C#

```
public bool Equals (long obj);
```

Parameters

obj Int64

An Int64 value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int64) method overloads. If an implicit conversion between the `obj` argument and an Int64 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int64) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int64 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, SByte, Int16, UInt16, Int32, and UInt32 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int64) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static long value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,15}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,17}", value.Equals(int1));
      TestObjectForEquality(int1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
```

```
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,15}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,17}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int64) = 112 (Byte): False
//
//       value = short1:            True
//       112 (Int64) = 112 (Int16): False
//
//       value = int1:              True
//       112 (Int64) = 112 (Int32): False
//
//       value = sbyte1:            True
//       112 (Int64) = 112 (SByte): False
//
//       value = ushort1:            True
//       112 (Int64) = 112 (UInt16): False
//
//       value = uint1:              True
//       112 (Int64) = 112 (UInt32): False
//
//       value = dec1:                False
//       112 (Int64) = 112 (Decimal): False
//
//       value = dbl1:               False
//       112 (Int64) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of an Int64 and equals the value of this instance; otherwise, `false`.

Examples

The following code example illustrates the use of `Equals` in the context of `Int64`, comparing two long values and returning `true` if they represent the same number, or `false` if they do not.

C#

```
Int64 myVariable1 = 80;
Int64 myVariable2 = 80;

// Get and display the declaring type.
Console.WriteLine("\nType of 'myVariable1' is '{0}' and"+
     " value is :{1}",myVariable1.GetType(), myVariable1);
Console.WriteLine("Type of 'myVariable2' is '{0}' and"+
     " value is :{1}",myVariable2.GetType(), myVariable2);

// Compare 'myVariable1' instance with 'myVariable2' Object.
if( myVariable1.Equals( myVariable2 ) )
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are equal");
else
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are not equal");
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Int64) method overloads. If an implicit conversion between the `obj` argument and an Int64 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(Int64) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not an Int64 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, SByte, Int16, UInt16, Int32, and UInt32 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Int64) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static long value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,15}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,15}", value.Equals(short1));
```

```
        TestObjectForEquality(short1);

        int int1 = 112;
        Console.WriteLine("value = int1: {0,17}", value.Equals(int1));
        TestObjectForEquality(int1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,15}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,15}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,17}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,20}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,19}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:            True
//       112 (Int64) = 112 (Byte): False
//
//       value = short1:            True
//       112 (Int64) = 112 (Int16): False
//
//       value = int1:              True
//       112 (Int64) = 112 (Int32): False
//
//       value = sbyte1:            True
//       112 (Int64) = 112 (SByte): False
//
//       value = ushort1:            True
//       112 (Int64) = 112 (UInt16): False
//
//       value = uint1:              True
//       112 (Int64) = 112 (UInt32): False
//
```

```
//       value = dec1:                 False
//       112 (Int64) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (Int64) = 112 (Double): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int64.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Int64.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Int64.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IsEvenInteger(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IsNegative(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IsOddInteger(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (long value);
```

Parameters

value Int64

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IsPositive(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IsPow2(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.LeadingZeroCount(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static long LeadingZeroCount (long value);
```

Parameters

`value` Int64

The value whose leading zeroes are to be counted.

Returns

Int64

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Log2(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static long Log2 (long value);
```

Parameters

`value` Int64

The value whose log2 is to be computed.

Returns

Int64

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Max(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static long Max (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.MaxMagnitude(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static long MaxMagnitude (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Min(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static long Min (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.MinMagnitude(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static long MinMagnitude (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 64-bit signed integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 64-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 64-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 64-bit signed integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.

```
C#

public static long Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

Returns

Int64

A 64-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int64.MinValue or greater than Int64.MaxValue.

-or-

`style` supports fractional digits, but `s` includes non-zero fractional digits.

Examples

The following example uses a variety of `style` and `provider` parameters to parse the string representations of Int64 values. It also illustrates some of the different ways the same string can be interpreted depending on the culture whose formatting information is used for the parsing operation.

C#

```
using System;
using System.Globalization;

public class ParseInt64
{
   public static void Main()
   {
      Convert("12,000", NumberStyles.Float | NumberStyles.AllowThousands,
              new CultureInfo("en-GB"));
      Convert("12,000", NumberStyles.Float | NumberStyles.AllowThousands,
              new CultureInfo("fr-FR"));
      Convert("12,000", NumberStyles.Float, new CultureInfo("en-US"));

      Convert("12 425,00", NumberStyles.Float |
NumberStyles.AllowThousands,
              new CultureInfo("sv-SE"));
      Convert("12,425.00", NumberStyles.Float |
NumberStyles.AllowThousands,
              NumberFormatInfo.InvariantInfo);
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowDecimalPoint,
              new CultureInfo("fr-FR"));
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowDecimalPoint,
              new CultureInfo("en-US"));
      Convert("631,900", NumberStyles.Integer |
NumberStyles.AllowThousands,
              new CultureInfo("en-US"));
   }
```

```
    private static void Convert(string value, NumberStyles style,
                                IFormatProvider provider)
    {
       try
       {
          long number = Int64.Parse(value, style, provider);
          Console.WriteLine("Converted '{0}' to {1}.", value, number);
       }
       catch (FormatException)
       {
          Console.WriteLine("Unable to convert '{0}'.", value);
       }
       catch (OverflowException)
       {
          Console.WriteLine("'{0}' is out of range of the Int64 type.",
value);
       }
    }
}
// This example displays the following output to the console:
//       Converted '12,000' to 12000.
//       Converted '12,000' to 12.
//       Unable to convert '12,000'.
//       Converted '12 425,00' to 12425.
//       Converted '12,425.00' to 12425.
//       '631,900' is out of range of the Int64 type.
//       Unable to convert '631,900'.
//       Converted '631,900' to 631900.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]exponential_digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag or at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9.
,	A culture-specific thousands separator symbol. The thousands separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, an OverflowException is thrown.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

Note

Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Int64 type. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements. However, *fractional-digits* must consist of only one or more 0 digits or an OverflowException is thrown.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The thousands separator (,) element.
NumberStyles.AllowCurrencySymbol	The *$* element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information used in parsing. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent.

C#

```
public static long Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information about `s`.

Returns

Int64

A 64-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static long Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int64

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its 64-bit signed integer equivalent.

```
C#
public static long Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Int64

A 64-bit signed integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to an Int64 value.

C#

```
protected void OkToLong_Click(object sender, EventArgs e)
{
    string locale;
    long number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
```

```
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Int64.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

This overload of the Parse(String, IFormatProvider) method is typically used to convert text that can be formatted in a variety of ways to an Int64 value. For example, it can be used to convert the text entered by a user into an HTML text box to a numeric value.

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional, and other items are as follows.

ws An optional white space.

sign An optional sign.

digits A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements that can be present in `s`, use the Int64.Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation, such as a NumberFormatInfo or CultureInfo object. The `provider` parameter supplies culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

Converts the string representation of a number to its 64-bit signed integer equivalent.

C#

```
public static long Parse (string s);
```

Parameters

s String

A string containing a number to convert.

Returns

Int64

A 64-bit signed integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than Int64.MinValue or greater than Int64.MaxValue.

Examples

The following example demonstrates how to convert a string value into a 64-bit signed integer value using the Int64.Parse(String) method. It then displays the resulting long integer value.

C#

```
using System;

public class ParseInt64
{
   public static void Main()
   {
      Convert("  179042  ");
      Convert(" -2041326 ");
      Convert(" +8091522 ");
      Convert("   1064.0   ");
      Convert("  178.3");
      Convert(String.Empty);
      Convert(((decimal) Int64.MaxValue) + 1.ToString());
   }

   private static void Convert(string value)
   {
      try
      {
         long number = Int64.Parse(value);
         Console.WriteLine("Converted '{0}' to {1}.", value, number);
      }
      catch (FormatException)
```

```
        {
            Console.WriteLine("Unable to convert '{0}'.", value);
        }
        catch (OverflowException)
        {
            Console.WriteLine("'{0}' is out of range.", value);
        }
    }
}
// This example displays the following output to the console:
//       Converted '   179042   ' to 179042.
//       Converted ' -2041326 ' to -2041326.
//       Converted ' +8091522 ' to 8091522.
//       Unable to convert '    1064.0    '.
//       Unable to convert '   178.3'.
//       Unable to convert ''.
//       '92233720368547758071' is out of range.
```

Remarks

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements that can be present in `s`, use either the Int64.Parse(String, NumberStyles) or the Int64.Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To parse a string using the formatting information of some other culture, use the Int64.Parse(String, NumberStyles, IFormatProvider) method.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#

public static long Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Int64

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static long Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Int64

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its 64-bit signed integer equivalent.

C#

```
public static long Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of NumberStyles values that indicates the permitted format of `s`. A typical value to specify is Integer.

Returns

Int64

A 64-bit signed integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than Int64.MinValue or greater than Int64.MaxValue.

-or-

`style` supports fractional digits but `s` includes non-zero fractional digits.

Examples

The following example uses the Int64.Parse(String, NumberStyles) method to parse the string representations of several Int64 values. The current culture for the example is en-US.

C#

```
using System;
using System.Globalization;

public class ParseInt32
{
   public static void Main()
   {
      Convert("104.0", NumberStyles.AllowDecimalPoint);
      Convert("104.9", NumberStyles.AllowDecimalPoint);
      Convert (" 106034", NumberStyles.None);
      Convert(" $17,198,064.42", NumberStyles.AllowCurrencySymbol |
                                 NumberStyles.Number);
      Convert(" $17,198,064.00", NumberStyles.AllowCurrencySymbol |
                                 NumberStyles.Number);
      Convert("103E06", NumberStyles.AllowExponent);
      Convert("1200E-02", NumberStyles.AllowExponent);
      Convert("1200E-03", NumberStyles.AllowExponent);
      Convert("-1,345,791", NumberStyles.AllowThousands);
      Convert("(1,345,791)", NumberStyles.AllowThousands |
                             NumberStyles.AllowParentheses);
      Convert("FFCA00A0", NumberStyles.HexNumber);
      Convert("0xFFCA00A0", NumberStyles.HexNumber);
   }

   private static void Convert(string value, NumberStyles style)
   {
      try
      {
         long number = Int64.Parse(value, style);
         Console.WriteLine("Converted '{0}' to {1}.", value, number);
      }
      catch (FormatException)
```

```
        {
            Console.WriteLine("Unable to convert '{0}'.", value);
        }
        catch (OverflowException)
        {
            Console.WriteLine("'{0}' is out of range of the Int64 type.",
value);
        }
    }
}
// The example displays the following output to the console:
//       Converted '104.0' to 104.
//       '104.9' is out of range of the Int64 type.
//       Unable to convert ' 106034'.
//       ' $17,198,064.42' is out of range of the Int64 type.
//       Converted ' $17,198,064.00' to 17198064.
//       Converted '103E06' to 103000000.
//       Converted '1200E-02' to 12.
//       '1200E-03' is out of range of the Int64 type.
//       Unable to convert '-1,345,791'.
//       Converted '(1,345,791)' to -1345791.
//       Converted 'FFCA00A0' to 4291428512.
//       Unable to convert '0xFFCA00A0'.
```

Remarks

The `style` parameter defines the style elements (such as white space, the positive or negative sign symbol, or the thousands separator symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may contain the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]exponential_digits][ws]

Or, if `style` includes AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can

Element	Description
	appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific thousands separator symbol. The current culture's thousands separator can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, an OverflowException is thrown.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Int64 type. Most of the remaining NumberStyles members control elements that may be but are not required to be

present in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation. If `s` represents a number in exponential notation, the resulting numeric value cannot include any non-zero fractional digits.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The *ws*, *sign*, thousands separator (,), and decimal point (.) elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be combined with the `s` parameter are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white space flags.)

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Int64.Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.PopCount(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static long PopCount (long value);
```

Parameters

`value` Int64

The value whose set bits are to be counted.

Returns

Int64

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.RotateLeft(Int64, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

C#

```
public static long RotateLeft (long value, int rotateAmount);
```

Parameters

`value` Int64

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

Int64

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.RotateRight(Int64, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static long RotateRight (long value, int rotateAmount);
```

Parameters

`value` Int64

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

Int64

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.Sign(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (long value);
```

Parameters

`value` Int64

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a minus sign if the value is negative, and a sequence of digits ranging from 0 to 9 with no leading zeroes.

Examples

The following example displays an Int64 value using the default ToString() method. It also displays the string representations of the Int64 value that results from using a number of standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
long value = -16325091;
// Display value using default ToString method.
Console.WriteLine(value.ToString());            // Displays -16325091
// Display value using some standard format specifiers.
Console.WriteLine(value.ToString("G"));         // Displays -16325091
Console.WriteLine(value.ToString("C"));         // Displays
($16,325,091.00)
Console.WriteLine(value.ToString("D"));         // Displays -16325091
Console.WriteLine(value.ToString("F"));         // Displays -16325091.00
Console.WriteLine(value.ToString("N"));         // Displays
-16,325,091.00
Console.WriteLine(value.ToString("N0"));         // Displays -16,325,091
Console.WriteLine(value.ToString("X"));         // Displays
FFFFFFFFFF06E61D
```

Remarks

The ToString() method formats an Int64 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An IFormatProvider that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example displays the string representation of an Int64 value using CultureInfo objects that represent several different cultures.

C#

```
long value = -16325901;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));
// This example displays the following output to the console:
//       -16325901
//       -16325901
//       -16325901
```

Remarks

The ToString(IFormatProvider) method formats an Int64 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)

To use format	For culture	Use the overload
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.
- A custom object that implements IFormatProvider and whose GetFormat method returns a NumberFormatInfo object that provides formatting information.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value is formatted with the NumberFormatInfo object for the current culture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid or not supported.

Examples

The following example displays an Int64 value using each of the supported standard numeric format specifiers together with two custom numeric format strings. In converting the numeric values to strings, the example uses the formatting conventions of the en-US culture.

C#

```
long value = -16325;
string specifier;

// Use standard numeric format specifier.
specifier = "G";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    G: -16325
specifier = "C";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    C: ($16,325.00)
specifier = "D8";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    D8: -00016325
specifier = "E4";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    E4: -1.6325E+004
specifier = "e3";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    e3: -1.633e+004
specifier = "F";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    F: -16325.00
specifier = "N";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    N: -16,325.00
specifier = "P";
Console.WriteLine("{0}: {1}", specifier,
(value/100000.0).ToString(specifier));
// Displays:    P: -16.33 %
specifier = "X";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    X: FFFFFFFFFFFFC03B

// Use custom numeric format specifiers.
specifier = "0,0.000";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:    0,0.000: -16,325.000
specifier = "#,#.00#;(#,#.00#)";
Console.WriteLine("{0}: {1}", specifier, (value*-1).ToString(specifier));
// Displays:    #,#.00#;(#,#.00#): 16,325.00
```

Remarks

The ToString(String) method formats an Int64 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid standard numeric format specifier except for "R", as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string (""), the return value of this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The return value of this instance is formatted with the NumberFormatInfo for the current culture.

See also

- Parse(String)
- String
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about this instance.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid or not supported.

Examples

The following example displays a positive and a negative value using each of the supported standard numeric format specifiers for three different cultures.

C#

```
// Define cultures whose formatting conventions are to be used.
CultureInfo[] cultures = {CultureInfo.CreateSpecificCulture("en-US"),
                          CultureInfo.CreateSpecificCulture("fr-FR"),
                          CultureInfo.CreateSpecificCulture("es-ES") };
long positiveNumber = 1679;
long negativeNumber = -3045;
string[] specifiers = {"G", "C", "D8", "E2", "F", "N", "N0", "P", "X8"};

foreach (string specifier in specifiers)
{
   foreach (CultureInfo culture in cultures)
   {
      // Display values with "G" format specifier.
      Console.WriteLine("{0} format using {1} culture: {2, 16} {3, 16}",
                        specifier, culture.Name,
                        positiveNumber.ToString(specifier, culture),
                        negativeNumber.ToString(specifier, culture));
   }
   Console.WriteLine();
}
// The example displays the following output to the console:
//    G format using en-US culture:             1679            -3045
//    G format using fr-FR culture:             1679            -3045
//    G format using es-ES culture:             1679            -3045
//
//    C format using en-US culture:        $1,679.00      ($3,045.00)
//    C format using fr-FR culture:       1 679,00 €      -3 045,00 €
//    C format using es-ES culture:       1.679,00 €      -3.045,00 €
//
//    D8 format using en-US culture:         00001679        -00003045
//    D8 format using fr-FR culture:         00001679        -00003045
//    D8 format using es-ES culture:         00001679        -00003045
//
//    E2 format using en-US culture:        1.68E+003       -3.05E+003
//    E2 format using fr-FR culture:        1,68E+003       -3,05E+003
//    E2 format using es-ES culture:        1,68E+003       -3,05E+003
//
//    F format using en-US culture:          1679.00         -3045.00
//    F format using fr-FR culture:          1679,00         -3045,00
//    F format using es-ES culture:          1679,00         -3045,00
//
//    N format using en-US culture:         1,679.00        -3,045.00
//    N format using fr-FR culture:         1 679,00        -3 045,00
//    N format using es-ES culture:         1.679,00        -3.045,00
//
//    N0 format using en-US culture:         1,679           -3,045
```

```
//    N0 format using fr-FR culture:         1 679           -3 045
//    N0 format using es-ES culture:         1.679           -3.045
//
//    P format using en-US culture:      167,900.00%     -304,500.00%
//    P format using fr-FR culture:     167 900,00 %    -304 500,00 %
//    P format using es-ES culture:     167.900,00 %    -304.500,00 %
//
//    X8 format using en-US culture:         0000068F FFFFFFFFFFFFF41B
//    X8 format using fr-FR culture:         0000068F FFFFFFFFFFFFF41B
//    X8 format using es-ES culture:         0000068F FFFFFFFFFFFFF41B
```

Remarks

The ToString(String, IFormatProvider) method formats an Int64 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be either a standard or a custom numeric format string. All standard numeric format strings other than "R" (or "r") are supported, as are all custom numeric format characters. If `format` is `null` or an empty string (""), the return value for this instance is formatted with the general numeric format specifier ("G").

The `provider` parameter is an object that implements the IFormatProvider interface. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string that is returned by this method. The object that implements IFormatProvider can be any of the following:

- A CultureInfo object that represents the culture whose formatting rules are to be used.
- A NumberFormatInfo object that contains specific numeric formatting information for this value.

- A custom object that implements IFormatProvider and whose GetFormat method returns a NumberFormatInfo object that provides formatting information.

If `provider` is `null` or a NumberFormatInfo object cannot be obtained from `provider`, the return value for this instance is formatted with the NumberFormatInfo for the current culture.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.

- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.TrailingZeroCount(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static long TrailingZeroCount (long value);
```

Parameters

`value` Int64

The value whose trailing zeroes are to be counted.

Returns

Int64

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current long number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current long number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, Int64)	Converts the span representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Int64)	Converts the string representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Int64)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, Int64)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Int64)	Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit signed integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int64)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Int64)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit

	signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Int64)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out long result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int64

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, Int64)

Converts the span representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out long result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`result` Int64

When this method returns, contains the 64-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Int64.MinValue or greater than Int64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Int64)

Converts the string representation of a number to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (string? s, out long result);
```

Parameters

s String

A string containing a number to convert.

result Int64

When this method returns, contains the 64-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than Int64.MinValue or greater than Int64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the Int64.TryParse(String, Int64) method with a number of different string values.

```
C#

using System;

public class StringParsing
{
   public static void Main()
   {
      TryToParse(null);
      TryToParse("160519");
      TryToParse("9432.0");
      TryToParse("16,667");
      TryToParse("   -322   ");
      TryToParse("+4302");
      TryToParse("(100);");
      TryToParse("01FA");
   }
```

```
    private static void TryToParse(string value)
    {
       bool success = Int64.TryParse(value, out long number);
       if (success)
       {
          Console.WriteLine("Converted '{0}' to {1}.", value, number);
       }
       else
       {
          if (value == null) value = "";
          Console.WriteLine("Attempted conversion of '{0}' failed.",
value);
       }
    }
}
// The example displays the following output to the console:
//       Attempted conversion of '' failed.
//       Converted '160519' to 160519.
//       Attempted conversion of '9432.0' failed.
//       Attempted conversion of '16,667' failed.
//       Converted '   -322   ' to -322.
//       Converted '+4302' to 4302.
//       Attempted conversion of '(100);' failed.
//       Attempted conversion of '01FA' failed.
```

Some of the strings that the TryParse(String, Int64) method is unable to convert in this example are:

- "9432.0". The conversion fails because the string cannot contain a decimal separator; it must contain integral digits only.
- "16,667". The conversion fails because the string cannot contain group separators; it must contain integral digits only.
- "(100)". The conversion fails because the string cannot contain a negative sign other than the one defined by the current culture's NumberFormatInfo.NegativeSign and NumberFormatInfo.NumberNegativePattern properties.
- "01FA". The conversion fails because the string cannot contain hexadecimal digits; it must contain decimal digits only.

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails. It eliminates the need to use exception

handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	An optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces together with a leading sign are allowed. To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the TryParse(String, NumberStyles, IFormatProvider, Int64) method.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see CurrentInfo.

This overload of the TryParse method interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the TryParse(String, NumberStyles, IFormatProvider, Int64) overload.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, Int64)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out long result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int64

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, Int64)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
long result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Int64

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Int64)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
long result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Int64

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Int64)

Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit signed integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out long
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` Int64

When this method returns, contains the 64-bit signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Int64)

Converts the span representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
long result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the characters representing the number to convert. The span is interpreted using the style specified by `style`.

style NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

provider IFormatProvider

An object that supplies culture-specific formatting information about `s`.

result Int64

When this method returns, contains the 64-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int64.MinValue or greater than Int64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Int64)

Converts the string representation of a number in a specified style and culture-specific format to its 64-bit signed integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out long result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Int64

When this method returns, contains the 64-bit signed integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than Int64.MinValue or greater than

Int64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, Int64) method with a number of different string and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class StringParsing
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "106779";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-30677";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);

      styles = NumberStyles.AllowLeadingSign;
      CallTryParse(numericString, styles);

      numericString = "301677-";
      CallTryParse(numericString, styles);
```

```
      styles = styles | NumberStyles.AllowTrailingSign;
      CallTryParse(numericString, styles);

      numericString = "$10634";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      styles = NumberStyles.Integer | NumberStyles.AllowCurrencySymbol;
      CallTryParse(numericString, styles);

      numericString = "10345.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "22,593";
      styles = NumberStyles.Integer | NumberStyles.AllowThousands;
      CallTryParse(numericString, styles);

      numericString = "12E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "12E03";
      CallTryParse(numericString, styles);

      numericString = "80c1";
      CallTryParse(numericString, NumberStyles.HexNumber);

      numericString = "0x80C1";
      CallTryParse(numericString, NumberStyles.HexNumber);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      CultureInfo provider;

      // If currency symbol is allowed, use en-US culture.
      if ((styles & NumberStyles.AllowCurrencySymbol) > 0)
         provider = new CultureInfo("en-US");
      else
         provider = CultureInfo.InvariantCulture;

      bool success = Int64.TryParse(stringToConvert, styles,
                                    provider, out long number);
      if (success)
         Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
      else
         Console.WriteLine($"Attempted conversion of '{stringToConvert}'
```

```
failed.");
    }
}
// The example displays the following output to the console:
//       Converted '106779' to 106779.
//       Attempted conversion of '-30677' failed.
//       Converted '-30677' to -30677.
//       Attempted conversion of '301677-' failed.
//       Converted '301677-' to -301677.
//       Attempted conversion of '$10634' failed.
//       Converted '$10634' to 10634.
//       Converted '10345.00' to 10345.
//       Attempted conversion of '10345.72' failed.
//       Converted '22,593' to 22593.
//       Attempted conversion of '12E-01' failed.
//       Converted '12E03' to 12000.
//       Converted '80c1' to 32961.
//       Attempted conversion of '0x80C1' failed.
```

Remarks

The TryParse method is like the Parse method, except the TryParse method does not throw an exception if the conversion fails. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`,the `s` parameter may include the following elements:

[ws][$][sign][digits,]digits[.fractional_digits][e[sign]exponential_digits][ws]

Or, if the `style` parameter includes NumberStyles.AllowHexSpecifier:

[ws]hexdigits[ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end

Element	Description
	of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. A sign symbol can appear in `s` if `style` includes the NumberStyles.AllowLeadingSign or NumberStyles.AllowTrailingSign flags.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific thousands separator. The thousands separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The 'e' or 'E' character, which indicates that the value is represented in exponential notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be but are not required to be present in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in s in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits or the method returns `false`.
NumberStyles.AllowExponent	The `s` parameter can also use exponential notation. The `s` parameter must represent an integer within the range of the Int64 data type without a non-zero fractional component.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign can appear before *digits*.
NumberStyles.AllowTrailingSign	A sign can appear after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The thousands separator (,) element.
NumberStyles.AllowCurrencySymbol	The *$* element.
NumberStyles.Currency	All elements. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
NumberStyles.Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
NumberStyles.Number	The *ws*, *sign*, thousands separator (,), and decimal point (.) elements.
NumberStyles.Any	All styles, except `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value without a prefix. For example, "C9AF3" parses successfully, but "0xC9AF3" does not. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, NumberStyles.HexNumber, that includes both white space flags.)

The `provider` parameter is an IFormatProvider implementation, such as a CultureInfo object or a NumberFormatInfo object, whose GetFormat method returns a NumberFormatInfo object. The NumberFormatInfo object provides culture-specific information about the format of `s`. If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

```
C#
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

```
C#
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.To Int64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, unchanged.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToSByte(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The type to which to convert this Int64 value.

`provider` IFormatProvider

An IFormatProvider implementation that provides information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt16(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt32(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Int64 instance is cast to an IConvertible interface. The recommended alternative is to

call the Convert.ToUInt64(Int64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.IAdditionOperators<Int64,Int64,Int64>.AdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static long IAdditionOperators<long,long,long>.operator + (long left, long
right);
```

Parameters

`left` Int64

The value to which `right` is added.

`right` Int64

The value that's added to `left`.

Returns

Int64

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IAdditionOperators<Int64,Int64,Int64>.CheckedAdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static long IAdditionOperators<long,long,long>.op_CheckedAddition (long
left, long right);
```

Parameters

`left` Int64

The value to which `right` is added.

`right` Int64

The value that's added to `left`.

Returns

Int64

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IAdditiveIdentity<Int64,Int64>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static long
System.Numerics.IAdditiveIdentity<System.Int64,System.Int64>.AdditiveIdentit
y { get; }
```

Property Value

Int64

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.IBinaryInteger<Int64>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<long>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.IBinaryInteger<Int64>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<long>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int64.IBinaryInteger<Int64>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<long>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out long value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int64.IBinaryInteger<Int64>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<long>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out long value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` Int64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Int64.IBinaryInteger<Int64>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<long>.TryWriteBigEndian (Span<byte> destination, out int
bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBinaryInteger<Int64>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<long>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBinaryNumber<Int64>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static long System.Numerics.IBinaryNumber<System.Int64>.AllBitsSet { get; }
```

Property Value

Int64

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBitwiseOperators<Int64,Int64,Int64>.BitwiseAndOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static long IBitwiseOperators<long,long,long>.operator & (long left, long
right);
```

Parameters

`left` Int64

The value to `and` with `right`.

`right` Int64

The value to `and` with `left`.

Returns

Int64

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBitwiseOperators<Int64,Int64,Int64>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static long IBitwiseOperators<long,long,long>.operator | (long left, long
right);
```

Parameters

`left` Int64

The value to `or` with `right`.

`right` Int64

The value to `or` with `left`.

Returns

Int64

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBitwiseOperators<Int64,Int64,Int64>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static long IBitwiseOperators<long,long,long>.operator ^ (long left, long
right);
```

Parameters

`left` Int64

The value to xor with `right`.

`right` Int64

The value to xor with `left`.

Returns

Int64

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IBitwiseOperators<Int64,Int64,Int64>.OnesComplement(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static long IBitwiseOperators<long,long,long>.operator ~ (long value);
```

Parameters

`value` Int64

The value for which to compute the ones-complement.

Returns

Int64

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IComparisonOperators<Int64,Int64,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<long,long,bool>.operator > (long left, long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IComparisonOperators<Int64,Int64,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<long,long,bool>.operator >= (long left,
long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IComparisonOperators<Int64,Int64,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<long,long,bool>.operator < (long left, long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IComparisonOperators<Int64,Int64,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<long,long,bool>.operator <= (long left,
long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IDecrementOperators<Int64>.CheckedDecrement(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static long IDecrementOperators<long>.op_CheckedDecrement (long value);
```

Parameters

`value` Int64

The value to decrement.

Returns

Int64

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IDecrementOperators<Int64>.Decrement(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static long IDecrementOperators<long>.operator -- (long value);
```

Parameters

`value` Int64

The value to decrement.

Returns

Int64

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IDivisionOperators<Int64,Int64,Int64>.DivisionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static long IDivisionOperators<long,long,long>.operator / (long left, long right);
```

Parameters

`left` Int64

The value that `right` divides.

`right` Int64

The value that divides `left`.

Returns

Int64

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IEqualityOperators<Int64,Int64,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<long,long,bool>.operator == (long left, long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IEqualityOperators<Int64,Int64,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<long,long,bool>.operator != (long left, long right);
```

Parameters

`left` Int64

The value to compare with `right`.

`right` Int64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IIncrementOperators<Int64>.CheckedIncrement(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static long IIncrementOperators<long>.op_CheckedIncrement (long value);
```

Parameters

`value` Int64

The value to increment.

Returns

Int64

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IIncrementOperators<Int64>.Increment(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static long IIncrementOperators<long>.operator ++ (long value);
```

Parameters

`value` Int64

The value to increment.

Returns

Int64

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IMinMaxValue<Int64>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static long System.Numerics.IMinMaxValue<System.Int64>.MaxValue { get; }
```

Property Value

Int64

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IMinMaxValue<Int64>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static long System.Numerics.IMinMaxValue<System.Int64>.MinValue { get; }
```

Property Value

Int64

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IModulusOperators<Int64,Int64,Int64>.Modulus(Int64, Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static long IModulusOperators<long,long,long>.operator % (long left, long
right);
```

Parameters

`left` Int64

The value that `right` divides.

`right` Int64

The value that divides `left`.

Returns

Int64

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IMultiplicativeIdentity<Int64,Int64>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static long
System.Numerics.IMultiplicativeIdentity<System.Int64,System.Int64>.MultiplicativeIdentity { get; }
```

Property Value

Int64

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.IMultiplyOperators<Int64,Int64,Int64>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static long IMultiplyOperators<long,long,long>.op_CheckedMultiply (long
left, long right);
```

Parameters

`left` Int64

The value that `right` multiplies.

`right` Int64

The value that multiplies `left`.

Returns

Int64

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IMultiplyOperators<Int64,Int64,Int64>.MultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static long IMultiplyOperators<long,long,long>.operator * (long left, long right);
```

Parameters

`left` Int64

The value that `right` multiplies.

`right` Int64

The value that multiplies `left`.

Returns

Int64

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumber<Int64>.MaxNumber(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static long INumber<long>.MaxNumber (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumber<Int64>.MinNumber(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static long INumber<long>.MinNumber (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsCanonical(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<long>.IsCanonical (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsComplexNumber(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<long>.IsComplexNumber (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsFinite(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<long>.IsFinite (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsImaginaryNumber(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

```
C#
static bool INumberBase<long>.IsImaginaryNumber (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsInfinity(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<long>.IsInfinity (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsInteger(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<long>.IsInteger (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsNaN(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<long>.IsNaN (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsNegativeInfinity(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<long>.IsNegativeInfinity (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsNormal(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<long>.IsNormal (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsPositiveInfinity(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<long>.IsPositiveInfinity (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsRealNumber(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<long>.IsRealNumber (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsSubnormal(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<long>.IsSubnormal (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.IsZero(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<long>.IsZero (long value);
```

Parameters

`value` Int64

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.MaxMagnitudeNumber(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#
static long INumberBase<long>.MaxMagnitudeNumber (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.MinMagnitudeNumber(Int64, Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static long INumberBase<long>.MinMagnitudeNumber (long x, long y);
```

Parameters

x Int64

The value to compare with `y`.

y Int64

The value to compare with `x`.

Returns

Int64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static long System.Numerics.INumberBase<System.Int64>.One { get; }
```

Property Value

Int64

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int64.INumberBase<Int64>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Int64>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<long>.TryConvertFromChecked<TOther> (TOther value,
out long result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.INumberBase<Int64>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<long>.TryConvertFromSaturating<TOther> (TOther
value, out long result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.INumberBase<Int64>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<long>.TryConvertFromTruncating<TOther> (TOther
value, out long result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Int64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Int64.INumberBase<Int64>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<long>.TryConvertToChecked<TOther> (long value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Int64.INumberBase<Int64>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<long>.TryConvertToSaturating<TOther> (long value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<long>.TryConvertToTruncating<TOther> (long value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Int64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.INumberBase<Int64>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static long System.Numerics.INumberBase<System.Int64>.Zero { get; }
```

Property Value

Int64

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int64.IShift Operators<Int64,Int32,Int64>.Left Shift(Int64, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static long IShiftOperators<long,int,long>.operator << (long value, int
shiftAmount);
```

Parameters

`value` Int64

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

Int64

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IShift Operators<Int64,Int32,Int64>.RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static long IShiftOperators<long,int,long>.operator >> (long value, int
shiftAmount);
```

Parameters

`value` Int64

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int64

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IShift Operators<Int64,Int32,Int64>.Unsigned RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static long IShiftOperators<long,int,long>.op_UnsignedRightShift (long
value, int shiftAmount);
```

Parameters

`value` Int64

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

Int64

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.ISignedNumber<Int64>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
static long System.Numerics.ISignedNumber<System.Int64>.NegativeOne { get; }
```

Property Value

Int64

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Int64.ISubtractionOperators<Int64,Int64,Int64>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static long ISubtractionOperators<long,long,long>.op_CheckedSubtraction
(long left, long right);
```

Parameters

`left` Int64

The value from which `right` is subtracted.

`right` Int64

The value that's subtracted from `left`.

Returns

Int64

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.ISubtractionOperators<Int64,Int64,Int64>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static long ISubtractionOperators<long,long,long>.operator - (long left,
long right);
```

Parameters

`left` Int64

The value from which `right` is subtracted.

`right` Int64

The value that's subtracted from `left`.

Returns

Int64

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IUnaryNegationOperators<Int64,Int64>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static long IUnaryNegationOperators<long,long>.op_CheckedUnaryNegation (long
value);
```

Parameters

`value` Int64

The value for which to compute the unary negation.

Returns

Int64

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IUnaryNegationOperators<Int64,Int64>.UnaryNegation(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static long IUnaryNegationOperators<long,long>.operator - (long value);
```

Parameters

`value` Int64

The value for which to compute the unary negation.

Returns

Int64

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Int64.IUnaryPlusOperators<Int64,Int64>.UnaryPlus(Int64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static long IUnaryPlusOperators<long,long>.operator + (long value);
```

Parameters

`value` Int64

The value for which to compute the unary plus.

Returns

Int64

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a signed integer where the bit-width is the same as a pointer.

C#

```
public readonly struct IntPtr : IComparable<IntPtr>, IEquatable<IntPtr>,
IParsable<IntPtr>, ISpanParsable<IntPtr>, IUtf8SpanParsable<IntPtr>,
System.Numerics.IAdditionOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.IAdditiveIdentity<IntPtr,IntPtr>,
System.Numerics.IBinaryInteger<IntPtr>,
System.Numerics.IBinaryNumber<IntPtr>,
System.Numerics.IBitwiseOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.IComparisonOperators<IntPtr,IntPtr,bool>,
System.Numerics.IDecrementOperators<IntPtr>,
System.Numerics.IDivisionOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.IEqualityOperators<IntPtr,IntPtr,bool>,
System.Numerics.IIncrementOperators<IntPtr>,
System.Numerics.IMinMaxValue<IntPtr>,
System.Numerics.IModulusOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.IMultiplicativeIdentity<IntPtr,IntPtr>,
System.Numerics.IMultiplyOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.INumber<IntPtr>, System.Numerics.INumberBase<IntPtr>,
System.Numerics.IShiftOperators<IntPtr,int,IntPtr>,
System.Numerics.ISignedNumber<IntPtr>,
System.Numerics.ISubtractionOperators<IntPtr,IntPtr,IntPtr>,
System.Numerics.IUnaryNegationOperators<IntPtr,IntPtr>,
System.Numerics.IUnaryPlusOperators<IntPtr,IntPtr>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → IntPtr

Implements IComparable , IComparable<IntPtr> , IEquatable<IntPtr> , IFormattable , ISerializable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<IntPtr> , IParsable<TSelf> , ISpanParsable<IntPtr> , ISpanParsable<TSelf> , IAdditionOperators<IntPtr,IntPtr,IntPtr> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<IntPtr,IntPtr> ,

IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<IntPtr> ,

IBinaryNumber<IntPtr> , IBinaryNumber<TSelf> ,

IBitwiseOperators<IntPtr,IntPtr,IntPtr> ,

IBitwiseOperators<TSelf,TSelf,TSelf> ,

IComparisonOperators<IntPtr,IntPtr,Boolean> ,

IComparisonOperators<TSelf,TSelf,Boolean> ,

IDecrementOperators<IntPtr> , IDecrementOperators<TSelf> ,

IDivisionOperators<IntPtr,IntPtr,IntPtr> ,

IDivisionOperators<TSelf,TSelf,TSelf> ,

IEqualityOperators<IntPtr,IntPtr,Boolean> ,

IEqualityOperators<TSelf,TOther,TResult> ,

IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<IntPtr> ,

IIncrementOperators<TSelf> , IMinMaxValue<IntPtr> ,

IModulusOperators<IntPtr,IntPtr,IntPtr> ,

IModulusOperators<TSelf,TSelf,TSelf> ,

IMultiplicativeIdentity<IntPtr,IntPtr> , IMultiplicativeIdentity<TSelf,TSelf> ,

IMultiplyOperators<IntPtr,IntPtr,IntPtr> ,

IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<IntPtr> ,

INumber<TSelf> , INumberBase<IntPtr> , INumberBase<TSelf> ,

IShiftOperators<IntPtr,Int32,IntPtr> , IShiftOperators<TSelf,Int32,TSelf> ,

ISignedNumber<IntPtr> , ISubtractionOperators<IntPtr,IntPtr,IntPtr> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

IUnaryNegationOperators<IntPtr,IntPtr> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<IntPtr,IntPtr> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<IntPtr> ,

IUtf8SpanParsable<TSelf>

Examples

The following example uses managed pointers to reverse the characters in an array. After it initializes a String object and gets its length, it does the following:

1. Calls the Marshal.StringToHGlobalAnsi method to copy the Unicode string to unmanaged memory as an ANSI (one-byte) character. The method returns an IntPtr object that points to the beginning of the unmanaged string. The Visual Basic example uses this pointer directly; in the C++, F# and C# examples, it is cast to a pointer to a byte.

2. Calls the Marshal.AllocHGlobal method to allocate the same number of bytes as the unmanaged string occupies. The method returns an IntPtr object that points to the beginning of the unmanaged block of memory. The Visual Basic example uses this pointer directly; in the C++, F# and C# examples, it is cast to a pointer to a byte.

3. The Visual Basic example defines a variable named `offset` that is equal to the length of the ANSI string. It is used to determine the offset into unmanaged memory to which the next character in the ANSI string is copied. Because its starting value is the length of the string, the copy operation will copy a character from the start of the string to the end of the memory block.

   The C#, F# and C++ examples call the ToPointer method to get an unmanaged pointer to the starting address of the string and the unmanaged block of memory, and they add one less than the length of the string to the starting address of the ANSI string. Because the unmanaged string pointer now points to the end of the string, the copy operation will copy a character from the end of the string to the start of the memory block.

4. Uses a loop to copy each character from the string to the unmanaged block of memory.

   The Visual Basic example calls the Marshal.ReadByte(IntPtr, Int32) method to read the byte (or one-byte character) at a specified offset from the managed pointer to the ANSI string. The offset is incremented with each iteration of the loop. It then calls the Marshal.WriteByte(IntPtr, Int32, Byte) method to write the byte to the memory address defined by the starting address of the unmanaged block of memory plus `offset`. It then decrements `offset`.

   The C#, F# and C++ examples perform the copy operation, then decrement the pointer to the address of the next location in the unmanaged ANSI string and increment the pointer to the next address in the unmanaged block.

5. All examples call the Marshal.PtrToStringAnsi to convert the unmanaged memory block containing the copied ANSI string to a managed Unicode String object.

6. After displaying the original and reversed strings, all examples call the FreeHGlobal method to free the memory allocated for the unmanaged ANSI string and the unmanaged block of memory.

C#

```
using System;
using System.Runtime.InteropServices;

class NotTooSafeStringReverse
{
    static public void Main()
    {
        string stringA = "I seem to be turned around!";
        int copylen = stringA.Length;

        // Allocate HGlobal memory for source and destination strings
        IntPtr sptr = Marshal.StringToHGlobalAnsi(stringA);
        IntPtr dptr = Marshal.AllocHGlobal(copylen + 1);

        // The unsafe section where byte pointers are used.
        unsafe
        {
            byte *src = (byte *)sptr.ToPointer();
            byte *dst = (byte *)dptr.ToPointer();

            if (copylen > 0)
            {
                // set the source pointer to the end of the string
                // to do a reverse copy.
                src += copylen - 1;

                while (copylen-- > 0)
                {
                    *dst++ = *src--;
                }
                *dst = 0;
            }
        }
        string stringB = Marshal.PtrToStringAnsi(dptr);

        Console.WriteLine("Original:\n{0}\n", stringA);
        Console.WriteLine("Reversed:\n{0}", stringB);

        // Free HGlobal memory
        Marshal.FreeHGlobal(dptr);
        Marshal.FreeHGlobal(sptr);
    }
}

// The progam has the following output:
//
// Original:
```

```
// I seem to be turned around!
//
// Reversed:
// !dnuora denrut eb ot mees I
```

Remarks

The IntPtr type is designed to be an integer whose size is the same as a pointer. That is, an instance of this type is expected to be 32 bits in a 32-bit process and 64 bits in a 64-bit process.

The IntPtr type can be used by languages that support pointers and as a common means of referring to data between languages that do and do not support pointers.

IntPtr objects can also be used to hold handles. For example, instances of IntPtr are used extensively in the System.IO.FileStream class to hold file handles.

> **Note**
>
> Using **IntPtr** as a pointer or a handle is error prone and unsafe. It is simply an integer type that can be used as an interchange format for pointers and handles due to being the same size. Outside of specific interchange requirements, such as for passing data to a language that doesn't support pointers, a correctly typed pointer should be used to represent pointers and **SafeHandle** should be used to represent handles.

This type implements the ISerializable. In .NET 5 and later versions, this type also implements the IFormattable interfaces. In .NET 7 and later versions, this type also implements the IBinaryInteger<TSelf>, IMinMaxValue<TSelf>, and ISignedNumber<TSelf> interfaces.

In C# starting from version 9.0, you can use the built-in `nint` type to define native-sized integers. This type is represented by the IntPtr type internally and provides operations and conversions that are appropriate for integer types. For more information, see nint and nuint types.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, `nint` is an alias for IntPtr in the same way that `int` is an alias for Int32.

Constructors

Expand table

IntPtr(Int32)	Initializes a new instance of IntPtr using the specified 32-bit signed integer.
IntPtr(Int64)	Initializes a new instance of IntPtr using the specified 64-bit signed integer.
Int Ptr(Void*)	Initializes a new instance of IntPtr using the specified pointer to an unspecified type.

Fields

Expand table

Zero	A read-only field that represents a signed integer that has been initialized to zero.

Properties

Expand table

MaxValue	Gets the largest possible value of IntPtr.
MinValue	Gets the smallest possible value of IntPtr.
Size	Gets the size of this instance.

Methods

Expand table

Abs(IntPtr)	Computes the absolute of a value.
Add(IntPtr, Int32)	Adds an offset to a signed integer.
Clamp(IntPtr, IntPtr, IntPtr)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(IntPtr)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the

	current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CopySign(IntPtr, IntPtr)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(IntPtr, IntPtr)	Computes the quotient and remainder of two values.
Equals(IntPtr)	Indicates whether the current object is equal to another object of the same type.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
IsEvenInteger(IntPtr)	Determines if a value represents an even integral number.
IsNegative(IntPtr)	Determines if a value is negative.
IsOddInteger(IntPtr)	Determines if a value represents an odd integral number.
IsPositive(IntPtr)	Determines if a value is positive.
IsPow2(IntPtr)	Determines if a value is a power of two.
LeadingZeroCount(IntPtr)	Computes the number of leading zeros in a value.
Log2(IntPtr)	Computes the log2 of a value.
Max(IntPtr, IntPtr)	Compares two values to compute which is greater.
MaxMagnitude(IntPtr, IntPtr)	Compares two values to compute which is greater.
Min(IntPtr, IntPtr)	Compares two values to compute which is lesser.
MinMagnitude(IntPtr, IntPtr)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer equivalent.
Parse(String)	Converts the string representation of a number to its signed native integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its signed native integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its signed native integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent.
PopCount(IntPtr)	Computes the number of bits that are set in a value.
RotateLeft(IntPtr, Int32)	Rotates a value left by a given amount.
RotateRight(IntPtr, Int32)	Rotates a value right by a given amount.
Sign(IntPtr)	Computes the sign of a value.
Subtract(IntPtr, Int32)	Subtracts an offset from a signed integer.
ToInt32()	Converts the value of this instance to a 32-bit signed integer.
ToInt64()	Converts the value of this instance to a 64-bit signed integer.
ToPointer()	Converts the value of this instance to a pointer to an unspecified type.
ToString()	Converts the numeric value of the current IntPtr object to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString(String)	Converts the numeric value of the current IntPtr object to its equivalent string representation.

ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
TrailingZeroCount(IntPtr)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, IntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, IntPtr)	Tries to convert a UTF-8 character span containing the string representation of a number to its signed integer equivalent.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, IntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, IntPtr)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, IntPtr)	Converts the read-only span of characters representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, IntPtr)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, IntPtr)	Tries to parse a string into a value.
TryParse(String, IntPtr)	Converts the string representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, NumberStyles, IFormatProvider, IntPtr)	Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent. A return value indicates whether the conversion succeeded.

Operators

Expand table

Addition(IntPtr, Int32)	Adds an offset to a signed integer.
Equality(IntPtr, IntPtr)	Determines whether two specified instances of IntPtr are equal.
Explicit(Int32 to IntPtr)	Converts the value of a 32-bit signed integer to an IntPtr.
Explicit(Int64 to IntPtr)	Converts the value of a 64-bit signed integer to an IntPtr.
Explicit(IntPtr to Int32)	Converts the value of the specified IntPtr to a 32-bit signed integer.
Explicit(IntPtr to Int64)	Converts the value of the specified IntPtr to a 64-bit signed integer.
Explicit(IntPtr to Void*)	Converts the value of the specified IntPtr to a pointer to an unspecified type. This API is not CLS-compliant.
Explicit(Void* to IntPtr)	Converts the specified pointer to an unspecified type to an IntPtr. This API is not CLS-compliant.
Inequality(IntPtr, IntPtr)	Determines whether two specified instances of IntPtr are not equal.
Subtraction(IntPtr, Int32)	Subtracts an offset from a signed integer.

Explicit Interface Implementations

Expand table

IAdditionOperators<IntPtr,IntPtr,IntPtr>.Addition(IntPtr, IntPtr)	Adds two values together to compute their sum.
IAdditionOperators<IntPtr,IntPtr,IntPtr>.CheckedAddition(IntPtr, IntPtr)	Adds two values together to compute their sum.
IAdditiveIdentity<IntPtr,IntPtr>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<IntPtr>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<IntPtr>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<IntPtr>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, IntPtr)	
IBinaryInteger<IntPtr>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, IntPtr)	

IBinaryInteger<IntPtr>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<IntPtr>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<IntPtr>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<IntPtr,IntPtr,IntPtr>.BitwiseAnd(IntPtr, IntPtr)	Computes the bitwise-and of two values.
IBitwiseOperators<IntPtr,IntPtr,IntPtr>.BitwiseOr(IntPtr, IntPtr)	Computes the bitwise-or of two values.
IBitwiseOperators<IntPtr,IntPtr,IntPtr>.ExclusiveOr(IntPtr, IntPtr)	Computes the exclusive-or of two values.
IBitwiseOperators<IntPtr,IntPtr,IntPtr>.OnesComplement(IntPtr)	Computes the ones-complement representation of a given value.
IComparisonOperators<IntPtr,IntPtr,Boolean>.GreaterThan(IntPtr, IntPtr)	Compares two values to determine which is greater.
IComparisonOperators<IntPtr,IntPtr,Boolean>.GreaterThanOrEqual(IntPtr, IntPtr)	Compares two values to determine which is greater or equal.
IComparisonOperators<IntPtr,IntPtr,Boolean>.LessThan(IntPtr, IntPtr)	Compares two values to determine which is less.
IComparisonOperators<IntPtr,IntPtr,Boolean>.LessThanOrEqual(IntPtr, IntPtr)	Compares two values to determine which is less or equal.
IDecrementOperators<IntPtr>.CheckedDecrement(IntPtr)	Decrements a value.
IDecrementOperators<IntPtr>.Decrement(IntPtr)	Decrements a value.
IDivisionOperators<IntPtr,IntPtr,IntPtr>.Division(IntPtr, IntPtr)	Divides one value by another to compute their quotient.
IIncrementOperators<IntPtr>.CheckedIncrement(IntPtr)	Increments a value.
IIncrementOperators<IntPtr>.Increment(IntPtr)	Increments a value.
IMinMaxValue<IntPtr>.MaxValue	Gets the maximum value of the current type.

IMinMaxValue<IntPtr>.MinValue	Gets the minimum value of the current type.
IModulusOperators<IntPtr,IntPtr,IntPtr>.Modulus(IntPtr, IntPtr)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<IntPtr,IntPtr>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
IMultiplyOperators<IntPtr,IntPtr,IntPtr>.CheckedMultiply(IntPtr, IntPtr)	Multiplies two values together to compute their product.
IMultiplyOperators<IntPtr,IntPtr,IntPtr>.Multiply(IntPtr, IntPtr)	Multiplies two values together to compute their product.
INumber<IntPtr>.MaxNumber(IntPtr, IntPtr)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<IntPtr>.MinNumber(IntPtr, IntPtr)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<IntPtr>.IsCanonical(IntPtr)	Determines if a value is in its canonical representation.
INumberBase<IntPtr>.IsComplexNumber(IntPtr)	Determines if a value represents a complex number.
INumberBase<IntPtr>.IsFinite(IntPtr)	Determines if a value is finite.
INumberBase<IntPtr>.IsImaginaryNumber(IntPtr)	Determines if a value represents a pure imaginary number.
INumberBase<IntPtr>.IsInfinity(IntPtr)	Determines if a value is infinite.
INumberBase<IntPtr>.IsInteger(IntPtr)	Determines if a value represents an integral number.
INumberBase<IntPtr>.IsNaN(IntPtr)	Determines if a value is NaN.
INumberBase<IntPtr>.IsNegativeInfinity(IntPtr)	Determines if a value is negative infinity.
INumberBase<IntPtr>.IsNormal(IntPtr)	Determines if a value is normal.
INumberBase<IntPtr>.IsPositiveInfinity(IntPtr)	Determines if a value is positive infinity.
INumberBase<IntPtr>.IsRealNumber(IntPtr)	Determines if a value represents a real number.
INumberBase<IntPtr>.IsSubnormal(IntPtr)	Determines if a value is subnormal.
INumberBase<IntPtr>.IsZero(IntPtr)	Determines if a value is zero.

INumberBase<IntPtr>.MaxMagnitudeNumber(IntPtr, IntPtr)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<IntPtr>.MinMagnitudeNumber(IntPtr, IntPtr)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<IntPtr>.One	Gets the value `1` for the type.
INumberBase<IntPtr>.Radix	Gets the radix, or base, for the type.
INumberBase<IntPtr>.TryConvertFromChecked<TOther>(TOther, IntPtr)	
INumberBase<IntPtr>.TryConvertFromSaturating<TOther>(TOther, IntPtr)	
INumberBase<IntPtr>.TryConvertFromTruncating<TOther>(TOther, IntPtr)	
INumberBase<IntPtr>.TryConvertToChecked<TOther>(IntPtr, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<IntPtr>.TryConvertToSaturating<TOther>(IntPtr, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<IntPtr>.TryConvertToTruncating<TOther>(IntPtr, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<IntPtr>.Zero	Gets the value `0` for the type.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data needed to serialize the current IntPtr object.
IShiftOperators<IntPtr,Int32,IntPtr>.LeftShift(IntPtr, Int32)	Shifts a value left by a given amount.
IShiftOperators<IntPtr,Int32,IntPtr>.RightShift(IntPtr, Int32)	Shifts a value right by a given amount.
IShiftOperators<IntPtr,Int32,IntPtr>.UnsignedRightShift(IntPtr, Int32)	Shifts a value right by a given amount.
ISignedNumber<IntPtr>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<IntPtr,IntPtr,IntPtr>.CheckedSubtraction(IntPtr, IntPtr)	Subtracts two values to compute their difference.
ISubtractionOperators<IntPtr,IntPtr,IntPtr>.Subtraction(IntPtr, IntPtr)	Subtracts two values to compute their difference.

IUnaryNegationOperators<IntPtr,IntPtr>.CheckedUnaryNegation(IntPtr)	Computes the checked unary negation of a value.
IUnaryNegationOperators<IntPtr,IntPtr>.UnaryNegation(IntPtr)	Computes the unary negation of a value.
IUnaryPlusOperators<IntPtr,IntPtr>.UnaryPlus(IntPtr)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

This type is thread safe.

See also

- UIntPtr

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of IntPtr.

Overloads

Expand table

IntPtr(Int32)	Initializes a new instance of IntPtr using the specified 32-bit signed integer.
IntPtr(Int64)	Initializes a new instance of IntPtr using the specified 64-bit signed integer.
IntPtr(Void*)	Initializes a new instance of IntPtr using the specified pointer to an unspecified type.

IntPtr(Int32)

Initializes a new instance of IntPtr using the specified 32-bit signed integer.

C#

```
public IntPtr (int value);
```

Parameters

`value` Int32

A 32-bit signed integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IntPtr(Int64)

Initializes a new instance of IntPtr using the specified 64-bit signed integer.

C#

```
public IntPtr (long value);
```

Parameters

`value` Int64

A 64-bit signed integer.

Exceptions

OverflowException

In a 32-bit process, `value` is too large or too small to represent as an IntPtr.

Remarks

An exception is only thrown if the value of `value` requires more bits than the executing process supports.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IntPtr(Void*)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of IntPtr using the specified pointer to an unspecified type.

C#

```
[System.CLSCompliant(false)]
public IntPtr (void* value);
```

Parameters

`value` Void*

A pointer to an unspecified type.

Attributes CLSCompliantAttribute

See also

- Void

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.Zero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A read-only field that represents a signed integer that has been initialized to zero.

```
C#

public static readonly IntPtr Zero;
```

Field Value

IntPtr

Remarks

The value of this field is not equivalent to `null`. Use this field to efficiently determine whether an instance of IntPtr has been set to a value other than zero.

For example, assume the variable, ip, is an instance of IntPtr. You can determine if it has been set by comparing it to the value returned by a constructor, for example: " `if ip != new IntPtr(0)...` ". However, invoking a constructor to get an uninitialized pointer is inefficient. It is better to code either " `if ip != IntPtr.Zero...` ", or " `if !IntPtr.Zero.Equals(ip)...` ".

When calling the Windows API from managed code, you can pass IntPtr.Zero instead of `null` if an argument is expected to be either a pointer or a `null`. For example, the following call to the Windows `CreateFile` function supplies IntPtr.Zero for the `pSecurityAttributes` and `hTemplateFile` argument values.

```
C#

using Microsoft.Win32.SafeHandles;
using System;
using System.Runtime.InteropServices;

public class Example
```

```
{
   private const uint GENERIC_READ = 0x80000000;
   private const uint OPEN_EXISTING = 3;
   private const uint FILE_ATTRIBUTE_NORMAL = 128;
   private const uint FILE_FLAG_OVERLAPPED = 0x40000000;

   [DllImport("kernel32.dll", SetLastError = true, CharSet =
CharSet.Unicode)]
   private static extern Microsoft.Win32.SafeHandles.SafeFileHandle
CreateFile(
            string lpFileName, System.UInt32 dwDesiredAccess, System.UInt32
dwShareMode,
            IntPtr pSecurityAttributes, System.UInt32 dwCreationDisposition,
            System.UInt32 dwFlagsAndAttributes, IntPtr hTemplateFile);

   public static void Main()
   {
      SafeFileHandle hnd = CreateFile("CallOfTheWild.txt", GENERIC_READ, 0,
                                      IntPtr.Zero, OPEN_EXISTING,
                                      FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_OVERLAPPED,
                                      IntPtr.Zero);
      if (hnd.IsInvalid) {
            Exception ex =
Marshal.GetExceptionForHR(Marshal.GetHRForLastWin32Error());
            Console.WriteLine("Attempt to open file failed:");
            Console.WriteLine("  {0}", ex.Message);
            return;
      }
      else {
         Console.WriteLine("File successfully opened.");
         hnd.Close();
      }
   }
}
// If the file cannot be found, the example displays the following output:
//    Attempt to open file failed:
//      The system cannot find the file specified. (Exception from HRESULT:
0x80070002)
```

> **Note**
>
> Although **Zero** is equivalent to `null` for Windows API functions with parameters or return values that can be either pointers or `null`, **Zero** is not equivalent to `null`. Passing `null` to the `IntPtr.Zero.Equals` method always returns `false`.

You can also test for a `null` return value from Windows API function calls that return either a pointer or a `null` by comparing the returned value with IntPtr.Zero. For example, the call to the `GetWindow` function in the following example tries to retrieve the

handle of a non-existent window. If it were called from unmanaged code, the function would return `null`, but when it is called from managed code, it returns IntPtr.Zero.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
   private const int GW_OWNER = 4;

   [DllImport("user32", CharSet=CharSet.Auto, SetLastError=true,
ExactSpelling=true)]
   public static extern IntPtr GetWindow(IntPtr hwnd, int wFlag);

   public static void Main()
   {
      IntPtr hwnd = new IntPtr(3);
      IntPtr hOwner = GetWindow(hwnd, GW_OWNER);
      if (hOwner == IntPtr.Zero)
         Console.WriteLine("Window not found.");
   }
}
// The example displays the following output:
//        Window not found.
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the largest possible value of IntPtr.

```
C#
public static IntPtr MaxValue { get; }
```

Property Value

IntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the smallest possible value of IntPtr.

```
C#
public static IntPtr MinValue { get; }
```

Property Value

IntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Size Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the size of this instance.

C#

```
public static int Size { get; }
```

Property Value

Int32

The size of a signed integer in the executing process, measured in bytes. The value of this property is 4 in a 32-bit process, and 8 in a 64-bit process. You can define the process type by setting the `/platform` switch when you compile your code with the C# and Visual Basic compilers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

IntPtr.Abs(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
public static IntPtr Abs (IntPtr value);
```

Parameters

`value` IntPtr

The value for which to get its absolute.

Returns

IntPtr

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Add(IntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds an offset to a signed integer.

C#

```
public static IntPtr Add (IntPtr pointer, int offset);
```

Parameters

pointer IntPtr

The signed integer to add the offset to.

offset Int32

The offset to add.

Returns

IntPtr

A new signed integer that reflects the addition of `offset` to `pointer`.

Examples

The following example instantiates an IntPtr object that points to the beginning of a ten-element array, and then calls the Add method to iterate the elements in the array.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
   public static void Main()
   {
```

```
        int[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
        unsafe {
           fixed(int* parr = arr) {
              IntPtr ptr = new IntPtr(parr);
              // Get the size of an array element.
              int size = sizeof(int);
              for (int ctr = 0; ctr < arr.Length; ctr++)
              {
                 IntPtr newPtr = IntPtr.Add(ptr, ctr * size);
                 Console.Write("{0}   ", Marshal.ReadInt32(newPtr));
              }
           }
        }
     }
  }
  // The example displays the following output:
  //       2   4   6   8   10   12   14   16   18   20
```

Remarks

The Add method does not throw an exception if the result is too large to represent as a signed integer in the executing process. Instead, the addition operation is performed in an unchecked context.

Languages that do not support operator overloading or custom operators can use this method to add an offset to the value of a pointer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Addition(IntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Clamp(IntPtr, IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static IntPtr Clamp (IntPtr value, IntPtr min, IntPtr max);
```

Parameters

`value` IntPtr

The value to clamp.

`min` IntPtr

The inclusive minimum to which `value` should clamp.

`max` IntPtr

The inclusive maximum to which `value` should clamp.

Returns

IntPtr

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(IntPtr)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

CompareTo(IntPtr)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (IntPtr value);
```

Parameters

`value` IntPtr

A signed native integer to compare.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

CompareTo(Object)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.CopySign(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

```
C#
public static IntPtr CopySign (IntPtr value, IntPtr sign);
```

Parameters

`value` IntPtr

The value whose magnitude is used in the result.

`sign` IntPtr

The value whose sign is used in the result.

Returns

IntPtr

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static IntPtr CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

IntPtr

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static IntPtr CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

IntPtr

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

```
C#

public static IntPtr CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

IntPtr

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.DivRem(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (IntPtr Quotient, IntPtr Remainder) DivRem (IntPtr left,
IntPtr right);
```

Parameters

`left` IntPtr

The value which `right` divides.

`right` IntPtr

The value which divides `left`.

Returns

ValueTuple<IntPtr,IntPtr>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(IntPtr)	Indicates whether the current object is equal to another object of the same type.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(IntPtr)

Indicates whether the current object is equal to another object of the same type.

C#

```
public bool Equals (IntPtr other);
```

Parameters

other IntPtr

An object to compare with this object.

Returns

Boolean

`true` if the current object is equal to `other`; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

```
C#
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance or `null`.

Returns

Boolean

`true` if `obj` is an instance of IntPtr and equals the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.IsEvenInteger(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IsNegative(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IsOddInteger(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IsPositive(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IsPow2(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.LeadingZeroCount(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static IntPtr LeadingZeroCount (IntPtr value);
```

Parameters

`value` IntPtr

The value whose leading zeroes are to be counted.

Returns

IntPtr

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Log2(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

```
C#
public static IntPtr Log2 (IntPtr value);
```

Parameters

`value` IntPtr

The value whose log2 is to be computed.

Returns

IntPtr

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Max(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static IntPtr Max (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.MaxMagnitude(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static IntPtr MaxMagnitude (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Min(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static IntPtr Min (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.MinMagnitude(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static IntPtr MinMagnitude (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its signed native integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its signed native integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Converts the string representation of a number to its signed native integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent.

```
C#

public static IntPtr Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

IntPtr

A signed native integer equivalent to the number contained in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer equivalent.

C#

```
public static IntPtr Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

IntPtr

A signed native integer equivalent to the number contained in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Exceptions

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static IntPtr Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

IntPtr

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its signed native integer equivalent.

C#

```
public static IntPtr Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

IntPtr

A signed native integer equivalent to the number contained in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its signed native integer equivalent.

```C#
public static IntPtr Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

s String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`.

Returns

IntPtr

A signed native integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#
```

```
public static IntPtr Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

IntPtr

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static IntPtr Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

IntPtr

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

Converts the string representation of a number to its signed native integer equivalent.

C#

```
public static IntPtr Parse (string s);
```

Parameters

`s` String

A string containing a number to convert.

Returns

IntPtr

A signed native integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.PopCount(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static IntPtr PopCount (IntPtr value);
```

Parameters

`value` IntPtr

The value whose set bits are to be counted.

Returns

IntPtr

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.RotateLeft(IntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static IntPtr RotateLeft (IntPtr value, int rotateAmount);
```

Parameters

`value` IntPtr

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

IntPtr

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.RotateRight(IntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static IntPtr RotateRight (IntPtr value, int rotateAmount);
```

Parameters

`value` IntPtr

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

IntPtr

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Sign(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (IntPtr value);
```

Parameters

`value` IntPtr

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.Subtract(IntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts an offset from a signed integer.

C#

```
public static IntPtr Subtract (IntPtr pointer, int offset);
```

Parameters

`pointer` IntPtr

The signed integer to subtract the offset from.

`offset` Int32

The offset to subtract.

Returns

IntPtr

A new signed integer that reflects the subtraction of `offset` from `pointer`.

Examples

The following example instantiates an IntPtr object that points to the end of a ten-element array, and then calls the Subtract method to iterate the elements in the array in reverse order.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
```

```
    public static void Main()
    {
       int[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
       // Get the size of a single array element.
       int size = sizeof(int);
       unsafe {
          fixed(int* pend = &arr[arr.GetUpperBound(0)]) {
             IntPtr ptr = new IntPtr(pend);
             for (int ctr = 0; ctr < arr.Length; ctr++)
             {
                IntPtr newPtr = IntPtr.Subtract(ptr, ctr * size);
                Console.Write("{0}   ", Marshal.ReadInt32(newPtr));
             }
          }
       }
    }
 }
 // The example displays the following output:
 //       20   18   16   14   12   10   8   6   4   2
```

Remarks

The Subtract method does not throw an exception if the result is too small to represent as a signed integer in the executing process. Instead, the subtraction operation is performed in an unchecked context.

Languages that do not support operator overloading or custom operators can use this method to subtract an offset from the value of a pointer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtraction(IntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.ToInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to a 32-bit signed integer.

C#

```
public int ToInt32 ();
```

Returns

Int32

A 32-bit signed integer equal to the value of this instance.

Exceptions

OverflowException

In a 64-bit process, the value of this instance is too large or too small to represent as a 32-bit signed integer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.ToInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to a 64-bit signed integer.

C#

```
public long ToInt64 ();
```

Returns

Int64

A 64-bit signed integer equal to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.ToPointer Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the value of this instance to a pointer to an unspecified type.

C#

```
[System.CLSCompliant(false)]
public void* ToPointer ();
```

Returns

Void*

A pointer to Void; that is, a pointer to memory containing data of an unspecified type.

Attributes CLSCompliantAttribute

Examples

The following example uses managed pointers to reverse the characters in an array. After it initializes a String object and gets its length, it does the following:

- Calls the Marshal.StringToHGlobalAnsi method to copy the Unicode string to unmanaged memory as ANSI (one-byte) characters. The method returns an IntPtr object that points to the beginning of the unmanaged string.
- Calls the Marshal.AllocHGlobal method to allocate the same number of bytes as the unmanaged string occupies. The method returns an IntPtr object that points to the beginning of the unmanaged block of memory.

- Calls the ToPointer method to get an unmanaged pointer to the starting address of the string and the unmanaged block of memory, and adds one less than the length of the string to the starting address of the ANSI string. Because the unmanaged string pointer now points to the end of the string, the copy operation will copy a character from the end of the string to the start of the memory block.

- Uses a loop to copy each character from the string to the unmanaged block of memory. After each copy operation, it decrements the pointer to the address of the next location in the unmanaged ANSI string and increments the pointer to the next address in the unmanaged block.

- Calls the Marshal.PtrToStringAnsi to convert the unmanaged memory block containing the copied ANSI string to a managed Unicode String object.

- After displaying the original and reversed strings, calls the Marshal.FreeHGlobal method to free the memory allocated for the unmanaged ANSI string and the unmanaged block of memory.

C#

```
using System;
using System.Runtime.InteropServices;

class NotTooSafeStringReverse
{
    static public void Main()
    {
        string stringA = "I seem to be turned around!";
        int copylen = stringA.Length;

        // Allocate HGlobal memory for source and destination strings
        IntPtr sptr = Marshal.StringToHGlobalAnsi(stringA);
        IntPtr dptr = Marshal.AllocHGlobal(copylen + 1);

        // The unsafe section where byte pointers are used.
        unsafe
        {
            byte *src = (byte *)sptr.ToPointer();
            byte *dst = (byte *)dptr.ToPointer();

            if (copylen > 0)
            {
                // set the source pointer to the end of the string
                // to do a reverse copy.
                src += copylen - 1;

                while (copylen-- > 0)
                {
                    *dst++ = *src--;
                }
```

```
                *dst = 0;
            }
        }
        string stringB = Marshal.PtrToStringAnsi(dptr);

        Console.WriteLine("Original:\n{0}\n", stringA);
        Console.WriteLine("Reversed:\n{0}", stringB);

        // Free HGlobal memory
        Marshal.FreeHGlobal(dptr);
        Marshal.FreeHGlobal(sptr);
    }
}

// The progam has the following output:
//
// Original:
// I seem to be turned around!
//
// Reversed:
// !dnuora denrut eb ot mees I
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of the current IntPtr object to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of the current IntPtr object to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString(String)	Converts the numeric value of the current IntPtr object to its equivalent string representation.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.

ToString()

Converts the numeric value of the current IntPtr object to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Remarks

If the value of the Size property for this instance is 4, then this method is equivalent to Int32.ToString; otherwise, this method is equivalent to Int64.ToString.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(String)

Converts the numeric value of the current IntPtr object to its equivalent string representation.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A format specification that governs how the current IntPtr object is converted.

Returns

String

The string representation of the value of the current IntPtr object.

Exceptions

FormatException

`format` is invalid or not supported.

Remarks

If the `format` parameter is `null` or an empty string ("), the return value is formatted with the general format specifier ("G"). For more information about numeric format specifiers, see the Standard Numeric Format Strings topic.

The return value is formatted using the invariant culture.

See also

- InvariantCulture
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Formats the value of the current instance using the specified format.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

The format to use. -or- A `null` reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.TrailingZeroCount(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static IntPtr TrailingZeroCount (IntPtr value);
```

Parameters

`value` IntPtr

The value whose trailing zeroes are to be counted.

Returns

IntPtr

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span where this instance's value formatted as a span of characters should be written.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

The characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IntPtr)	Tries to convert a UTF-8 character span containing the string representation of a number to its signed integer equivalent.
TryParse(ReadOnlySpan<Char>, IntPtr)	Converts the read-only span of characters representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, IntPtr)	Converts the string representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, IntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, IntPtr)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, IntPtr)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, IntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(String, NumberStyles, IFormatProvider, IntPtr)	Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, IntPtr)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer

	equivalent. A return value indicates whether the conversion succeeded.

TryParse(ReadOnlySpan<Byte>, IntPtr)

Tries to convert a UTF-8 character span containing the string representation of a number to its signed integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out IntPtr
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` IntPtr

When this method returns, contains the signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IntPtr)

Converts the read-only span of characters representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out IntPtr result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`result` IntPtr

When this method returns, contains the signed native integer equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(String, IntPtr)

Converts the string representation of a number to its signed native integer equivalent. A return value indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse (string? s, out IntPtr result);
```

Parameters

`s` String

A string containing a number to convert.

`result` IntPtr

When this method returns, contains the signed native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(ReadOnlySpan<Byte>, IFormatProvider, IntPtr)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out IntPtr result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` IntPtr

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, IntPtr)

Tries to parse a string into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out IntPtr result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` IntPtr

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, IntPtr)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
IntPtr result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` IntPtr

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, IntPtr)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
IntPtr result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` IntPtr

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, NumberStyles, IFormatProvider, IntPtr)

Converts the string representation of a number in a specified style and culture-specific format to its signed native integer equivalent. A return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out IntPtr result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` IntPtr

When this method returns, contains the signed native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, IntPtr)

Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its signed native integer equivalent. A return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
IntPtr result);
```

Parameters

s ReadOnlySpan<Char>

A read-only span of characters containing a number to convert. The string is interpreted using the style specified by `style`.

style NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` IntPtr

When this method returns, contains the signed native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.Addition(IntPtr, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds an offset to a signed integer.

```
C#

public static IntPtr operator + (IntPtr pointer, int offset);
```

Parameters

`pointer` IntPtr

The signed integer to add the offset to.

`offset` Int32

The offset to add.

Returns

IntPtr

A new signed integer that reflects the addition of `offset` to `pointer`.

Remarks

The Addition method defines the addition operation for IntPtr objects. It enables code such as the following.

```
C#

int[] arr = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
unsafe {
   fixed(int* parr = arr)
   {
       IntPtr ptr = new IntPtr(parr);
       for (int ctr = 0; ctr < arr.Length; ctr++)
       {
```

```
            IntPtr newPtr = ptr + ctr * sizeof(Int32);
            Console.Write("{0}   ", Marshal.ReadInt32(newPtr));
        }
    }
}
// The example displays the following output:
//       2   4   6   8   10   12   14   16   18   20
```

Languages that do not support custom operators can call the Add method instead.

The addition operation does not throw an exception if the result is too large to represent as a signed integer in the executing process. Instead, it is performed in an unchecked context.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, this API is only accessible via reflection. The addition operator is directly recognized by the language and will follow the normal language behavior for addition operations, including overflowing in a `checked` context if the result is too large to represent.

The equivalent method for this operator is IntPtr.Add(IntPtr, Int32)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(IntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.Equality(IntPtr, IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of IntPtr are equal.

```
C#
public static bool operator == (IntPtr value1, IntPtr value2);
```

Parameters

`value1` IntPtr

The first signed integer to compare.

`value2` IntPtr

The second signed integer to compare.

Returns

Boolean

`true` if `value1` equals `value2`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Remarks

The equivalent method for this operator is IntPtr.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified data type to an IntPtr value, or an IntPtr value to a specified data type.

Overloads

Expand table

Explicit(Void* to IntPtr)	Converts the specified pointer to an unspecified type to an IntPtr. This API is not CLS-compliant.
Explicit(IntPtr to Int64)	Converts the value of the specified IntPtr to a 64-bit signed integer.
Explicit(IntPtr to Void*)	Converts the value of the specified IntPtr to a pointer to an unspecified type. This API is not CLS-compliant.
Explicit(Int64 to IntPtr)	Converts the value of a 64-bit signed integer to an IntPtr.
Explicit(IntPtr to Int32)	Converts the value of the specified IntPtr to a 32-bit signed integer.
Explicit(Int32 to IntPtr)	Converts the value of a 32-bit signed integer to an IntPtr.

Explicit(Void* to IntPtr)

> Important
>
> This API is not CLS-compliant.

Converts the specified pointer to an unspecified type to an IntPtr.

This API is not CLS-compliant.

```
C#

[System.CLSCompliant(false)]
public static explicit operator IntPtr (void* value);
```

Parameters

`value` Void*

A pointer to an unspecified type.

Returns

IntPtr

A new instance of IntPtr initialized to `value`.

Attributes CLSCompliantAttribute

See also

- IntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(IntPtr to Int64)

Converts the value of the specified IntPtr to a 64-bit signed integer.

```
C#
public static explicit operator long (IntPtr value);
```

Parameters

`value` IntPtr

The signed integer to convert.

Returns

Int64

The contents of `value`.

See also

- ToInt64()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(IntPtr to Void*)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified IntPtr to a pointer to an unspecified type.

This API is not CLS-compliant.

```
C#

[System.CLSCompliant(false)]
public static explicit operator void* (IntPtr value);
```

Parameters

`value` IntPtr

The signed integer to convert.

Returns

Void*

The contents of `value`.

Attributes CLSCompliantAttribute

See also

- ToPointer()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Int64 to IntPtr)

Converts the value of a 64-bit signed integer to an IntPtr.

```
C#
public static explicit operator IntPtr (long value);
```

Parameters

`value` Int64

A 64-bit signed integer.

Returns

IntPtr

A new instance of IntPtr initialized to `value`.

Exceptions

OverflowException

In a 32-bit process, `value` is too large to represent as an IntPtr.

See also

- IntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(IntPtr to Int32)

Converts the value of the specified IntPtr to a 32-bit signed integer.

```
C#
public static explicit operator int (IntPtr value);
```

Parameters

`value` IntPtr

The signed integer to convert.

Returns

Int32

The contents of `value`.

Exceptions

OverflowException

In a 64-bit process, the value of `value` is too large to represent as a 32-bit signed integer.

Remarks

An exception is only thrown if the value of `value` requires more bits than the executing process supports.

See also

- ToInt32()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Int32 to IntPtr)

Converts the value of a 32-bit signed integer to an IntPtr.

C#

```
public static explicit operator IntPtr (int value);
```

Parameters

`value` Int32

A 32-bit signed integer.

Returns

IntPtr

A new instance of IntPtr initialized to `value`.

See also

- IntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.Inequality(IntPtr, IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of IntPtr are not equal.

```
C#
public static bool operator != (IntPtr value1, IntPtr value2);
```

Parameters

`value1` IntPtr

The first signed integer to compare.

`value2` IntPtr

The second signed integer to compare.

Returns

Boolean

`true` if `value1` does not equal `value2`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Remarks

The equivalent method for this operator is IntPtr.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.Subtraction(IntPtr, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts an offset from a signed integer.

```
C#

public static IntPtr operator - (IntPtr pointer, int offset);
```

Parameters

`pointer` IntPtr

The signed integer to subtract the offset from.

`offset` Int32

The offset to subtract.

Returns

IntPtr

A new signed integer that reflects the subtraction of `offset` from `pointer`.

Remarks

The Subtraction method defines the subtraction operation for IntPtr objects. It enables code such as the following.

```
C#

int[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
unsafe {
   fixed(int* parr = &arr[arr.GetUpperBound(0)])
   {
      IntPtr ptr = new IntPtr(parr);
      for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
      {
```

```
            IntPtr newPtr = ptr - ctr * sizeof(Int32);
            Console.Write("{0}   ", Marshal.ReadInt32(newPtr));
        }
    }
}
// The example displays the following output:
//       20   18   16   14   12   10   8   6   4   2
```

Languages that do not support custom operators can call the Subtract method instead.

The subtraction operation does not throw an exception if the result is too small to represent as a signed integer in the executing process. Instead, it is performed in an unchecked context.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, this API is only accessible via reflection. The subtraction operator is directly recognized by the language and will follow the normal language behavior for subtraction operations, including overflowing in a `checked` context if the result is too small to represent.

The equivalent method for this operator is IntPtr.Subtract(IntPtr, Int32)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(IntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.IAdditionOperators<IntPtr,IntPtr,IntPtr>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static IntPtr IAdditionOperators<IntPtr,IntPtr,IntPtr>.operator + (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to which `right` is added.

`right` IntPtr

The value that's added to `left`.

Returns

IntPtr

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IAdditionOperators<IntPtr,IntPtr,IntPtr>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static IntPtr IAdditionOperators<IntPtr,IntPtr,IntPtr>.op_CheckedAddition
(IntPtr left, IntPtr right);
```

Parameters

`left` IntPtr

The value to which `right` is added.

`right` IntPtr

The value that's added to `left`.

Returns

IntPtr

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IAdditiveIdentity<IntPtr,IntPtr>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static IntPtr
System.Numerics.IAdditiveIdentity<System.IntPtr,System.IntPtr>.AdditiveIdentity { get; }
```

Property Value

IntPtr

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<IntPtr>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<IntPtr>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<IntPtr>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out IntPtr value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` IntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<IntPtr>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out IntPtr value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` IntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<IntPtr>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IBinaryInteger<IntPtr>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<IntPtr>.TryWriteLittleEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IBinaryNumber<IntPtr>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static IntPtr System.Numerics.IBinaryNumber<System.IntPtr>.AllBitsSet { get;
}
```

Property Value

IntPtr

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.IBitwiseOperators<IntPtr,IntPtr,IntPtr>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static IntPtr IBitwiseOperators<IntPtr,IntPtr,IntPtr>.operator & (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to `and` with `right`.

`right` IntPtr

The value to `and` with `left`.

Returns

IntPtr

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IBitwiseOperators<IntPtr,IntPtr,IntPtr>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static IntPtr IBitwiseOperators<IntPtr,IntPtr,IntPtr>.operator | (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to `or` with `right`.

`right` IntPtr

The value to `or` with `left`.

Returns

IntPtr

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IBitwiseOperators<IntPtr,IntPtr,IntPtr>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static IntPtr IBitwiseOperators<IntPtr,IntPtr,IntPtr>.operator ^ (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to xor with `right`.

`right` IntPtr

The value to xor with `left`.

Returns

IntPtr

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IBitwiseOperators<IntPtr,IntPtr,IntPtr>.OnesComplement Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#

static IntPtr IBitwiseOperators<IntPtr,IntPtr,IntPtr>.operator ~ (IntPtr
value);
```

Parameters

`value` IntPtr

The value for which to compute the ones-complement.

Returns

IntPtr

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IComparisonOperators<IntPtr,IntPtr,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<IntPtr,IntPtr,bool>.operator > (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to compare with `right`.

`right` IntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IComparisonOperators<IntPtr,IntPtr,Boolean>.GreaterThanOrEqualOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<IntPtr,IntPtr,bool>.operator >= (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to compare with `right`.

`right` IntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IComparisonOperators<IntPtr,IntPtr,Boolean>.LessThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<IntPtr,IntPtr,bool>.operator < (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to compare with `right`.

`right` IntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IComparisonOperators<IntPtr,IntPtr,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<IntPtr,IntPtr,bool>.operator <= (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value to compare with `right`.

`right` IntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IDecrementOperators<IntPtr>.CheckedDecrement(IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#

static IntPtr IDecrementOperators<IntPtr>.op_CheckedDecrement (IntPtr
value);
```

Parameters

`value` IntPtr

The value to decrement.

Returns

IntPtr

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IDecrementOperators<IntPtr>.Decrement(IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static IntPtr IDecrementOperators<IntPtr>.operator -- (IntPtr value);
```

Parameters

`value` IntPtr

The value to decrement.

Returns

IntPtr

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IDivisionOperators<IntPtr,IntPtr,IntPtr>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static IntPtr IDivisionOperators<IntPtr,IntPtr,IntPtr>.operator / (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value that `right` divides.

`right` IntPtr

The value that divides `left`.

Returns

IntPtr

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IIncrementOperators<IntPtr>.CheckedIncrement(IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#

static IntPtr IIncrementOperators<IntPtr>.op_CheckedIncrement (IntPtr
value);
```

Parameters

`value` IntPtr

The value to increment.

Returns

IntPtr

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IIncrementOperators<IntPtr>.Increment(IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static IntPtr IIncrementOperators<IntPtr>.operator ++ (IntPtr value);
```

Parameters

`value` IntPtr

The value to increment.

Returns

IntPtr

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IMinMaxValue<IntPtr>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static IntPtr System.Numerics.IMinMaxValue<System.IntPtr>.MaxValue { get; }
```

Property Value

IntPtr

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IMinMaxValue<IntPtr>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static IntPtr System.Numerics.IMinMaxValue<System.IntPtr>.MinValue { get; }
```

Property Value

IntPtr

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IModulusOperators<IntPtr,IntPtr,IntPtr>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static IntPtr IModulusOperators<IntPtr,IntPtr,IntPtr>.operator % (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value that `right` divides.

`right` IntPtr

The value that divides `left`.

Returns

IntPtr

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IMultiplicativeIdentity<IntPtr,IntPtr>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static IntPtr
System.Numerics.IMultiplicativeIdentity<System.IntPtr,System.IntPtr>.Multipl
icativeIdentity { get; }
```

Property Value

IntPtr

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

IntPtr.IMultiplyOperators<IntPtr,IntPtr,IntPtr>.CheckedMultiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

C#

```
static IntPtr IMultiplyOperators<IntPtr,IntPtr,IntPtr>.op_CheckedMultiply
(IntPtr left, IntPtr right);
```

Parameters

left IntPtr

The value that `right` multiplies.

right IntPtr

The value that multiplies `left`.

Returns

IntPtr

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IMultiplyOperators<IntPtr,IntPtr,IntPtr>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static IntPtr IMultiplyOperators<IntPtr,IntPtr,IntPtr>.operator * (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value that `right` multiplies.

`right` IntPtr

The value that multiplies `left`.

Returns

IntPtr

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumber<IntPtr>.MaxNumber(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static IntPtr INumber<IntPtr>.MaxNumber (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumber<IntPtr>.MinNumber(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static IntPtr INumber<IntPtr>.MinNumber (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsCanonical(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<IntPtr>.IsCanonical (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsComplexNumber(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<IntPtr>.IsComplexNumber (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsFinite(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

C#

```
static bool INumberBase<IntPtr>.IsFinite (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsImaginaryNumber(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<IntPtr>.IsImaginaryNumber (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsInfinity(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<IntPtr>.IsInfinity (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsInteger(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<IntPtr>.IsInteger (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsNaN(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<IntPtr>.IsNaN (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsNegativeInfinity(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<IntPtr>.IsNegativeInfinity (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsNormal(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<IntPtr>.IsNormal (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsPositiveInfinity(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<IntPtr>.IsPositiveInfinity (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsRealNumber(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<IntPtr>.IsRealNumber (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsSubnormal(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<IntPtr>.IsSubnormal (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.IsZero(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<IntPtr>.IsZero (IntPtr value);
```

Parameters

`value` IntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.MaxMagnitudeNumber(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#
static IntPtr INumberBase<IntPtr>.MaxMagnitudeNumber (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.MinMagnitudeNumber(IntPtr, IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static IntPtr INumberBase<IntPtr>.MinMagnitudeNumber (IntPtr x, IntPtr y);
```

Parameters

x IntPtr

The value to compare with `y`.

y IntPtr

The value to compare with `x`.

Returns

IntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static IntPtr System.Numerics.INumberBase<System.IntPtr>.One { get; }
```

Property Value

IntPtr

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.INumberBase<IntPtr>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.IntPtr>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<IntPtr>.TryConvertFromChecked<TOther> (TOther value,
out IntPtr result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` IntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<IntPtr>.TryConvertFromSaturating<TOther> (TOther
value, out IntPtr result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` IntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<IntPtr>.TryConvertFromTruncating<TOther> (TOther
value, out IntPtr result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` IntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<IntPtr>.TryConvertToChecked<TOther> (IntPtr value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` IntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<IntPtr>.TryConvertToSaturating<TOther> (IntPtr
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` IntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<IntPtr>.TryConvertToTruncating<TOther> (IntPtr
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` IntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.INumberBase<IntPtr>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static IntPtr System.Numerics.INumberBase<System.IntPtr>.Zero { get; }
```

Property Value

IntPtr

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.IShiftOperators<IntPtr,Int32,IntPtr>.LeftShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

C#

```
static IntPtr IShiftOperators<IntPtr,int,IntPtr>.operator << (IntPtr value,
int shiftAmount);
```

Parameters

`value` IntPtr

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

IntPtr

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IShiftOperators<IntPtr,Int32,IntPtr>.RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static IntPtr IShiftOperators<IntPtr,int,IntPtr>.operator >> (IntPtr value,
int shiftAmount);
```

Parameters

`value` IntPtr

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

IntPtr

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IShiftOperators<IntPtr,Int32,IntPtr>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static IntPtr IShiftOperators<IntPtr,int,IntPtr>.op_UnsignedRightShift
(IntPtr value, int shiftAmount);
```

Parameters

value IntPtr

The value that is shifted right by `shiftAmount`.

shiftAmount Int32

The amount by which `value` is shifted right.

Returns

IntPtr

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.ISignedNumber<IntPtr>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

```
C#

static IntPtr System.Numerics.ISignedNumber<System.IntPtr>.NegativeOne {
get; }
```

Property Value

IntPtr

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

IntPtr.ISubtractionOperators<IntPtr,IntPtr,IntPtr>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static IntPtr
ISubtractionOperators<IntPtr,IntPtr,IntPtr>.op_CheckedSubtraction (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value from which `right` is subtracted.

`right` IntPtr

The value that's subtracted from `left`.

Returns

IntPtr

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.ISubtractionOperators<IntPtr,IntPtr,IntPtr>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static IntPtr ISubtractionOperators<IntPtr,IntPtr,IntPtr>.operator - (IntPtr
left, IntPtr right);
```

Parameters

`left` IntPtr

The value from which `right` is subtracted.

`right` IntPtr

The value that's subtracted from `left`.

Returns

IntPtr

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IUnaryNegationOperators<IntPtr,IntPtr>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static IntPtr IUnaryNegationOperators<IntPtr,IntPtr>.op_CheckedUnaryNegation
(IntPtr value);
```

Parameters

`value` IntPtr

The value for which to compute the unary negation.

Returns

IntPtr

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IUnaryNegationOperators<IntPtr,IntPtr>.UnaryNegation Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#

static IntPtr IUnaryNegationOperators<IntPtr,IntPtr>.operator - (IntPtr
value);
```

Parameters

`value` IntPtr

The value for which to compute the unary negation.

Returns

IntPtr

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.IUnaryPlusOperators<IntPtr,IntPtr>.UnaryPlus(IntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

C#

```
static IntPtr IUnaryPlusOperators<IntPtr,IntPtr>.operator + (IntPtr value);
```

Parameters

`value` IntPtr

The value for which to compute the unary plus.

Returns

IntPtr

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IntPtr.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data needed to serialize the current IntPtr object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo object to populate with data.

`context` StreamingContext

The destination for this serialization. (This parameter is not used; specify `null`.)

Implements

GetObjectData(SerializationInfo, StreamingContext)

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

This method populates the `info` parameter with the value of the current IntPtr object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InvalidCastException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown for invalid casting or explicit conversion.

C#

```
public class InvalidCastException : SystemException
```

Inheritance Object → Exception → SystemException → InvalidCastException

Remarks

For more information about this API, see Supplemental API remarks for InvalidCastException.

Constructors

Expand table

InvalidCastException()	Initializes a new instance of the InvalidCastException class.
InvalidCastException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidCastException class with serialized data.
InvalidCastException(String)	Initializes a new instance of the InvalidCastException class with a specified error message.
InvalidCastException(String, Exception)	Initializes a new instance of the InvalidCastException class with a specified error message and a reference to the inner exception that is the cause of this exception.
InvalidCastException(String, Int32)	Initializes a new instance of the InvalidCastException class with a specified message and error code.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions
- Type Conversion in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

InvalidCastException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InvalidCastException class.

Overloads

Expand table

InvalidCastException()	Initializes a new instance of the InvalidCastException class.
InvalidCastException(String)	Initializes a new instance of the InvalidCastException class with a specified error message.
InvalidCastException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidCastException class with serialized data.
InvalidCastException(String, Exception)	Initializes a new instance of the InvalidCastException class with a specified error message and a reference to the inner exception that is the cause of this exception.
InvalidCastException(String, Int32)	Initializes a new instance of the InvalidCastException class with a specified message and error code.

InvalidCastException()

Initializes a new instance of the InvalidCastException class.

C#

```
public InvalidCastException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Cannot cast from source type to destination type." This message takes into account the current system culture.

The following table shows the initial property values for an instance of InvalidCastException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidCastException(String)

Initializes a new instance of the InvalidCastException class with a specified error message.

C#

```
public InvalidCastException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of InvalidCastException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidCastException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the InvalidCastException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected InvalidCastException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InvalidCastException(String, Exception)

Initializes a new instance of the InvalidCastException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public InvalidCastException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or returns `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of InvalidCastException.

Expand table

Property	Value
InnerException	The inner exception reference.

Property	Value
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidCastException(String, Int32)

Initializes a new instance of the InvalidCastException class with a specified message and error code.

C#

```
public InvalidCastException (string? message, int errorCode);
```

Parameters

`message` String

The message that indicates the reason the exception occurred.

`errorCode` Int32

The error code (HRESULT) value associated with the exception.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

This constructor supplies an HRESULT value that is accessible to inheritors of the InvalidCastException class, via the protected HResult property of the Exception class.

The following table shows the initial property values for an instance of InvalidCastException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

InvalidOperationException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a method call is invalid for the object's current state.

C#

```
public class InvalidOperationException : SystemException
```

Inheritance Object → Exception → SystemException → InvalidOperationException

Derived System.Data.Linq.DuplicateKeyException
System.Data.Linq.ForeignKeyReferenceAlreadyHasValueException
System.Data.Services.Client.DataServiceClientException
System.Data.Services.Client.DataServiceQueryException
System.Data.Services.Client.DataServiceRequestException
More...

Remarks

For more information about this API, see Supplemental API remarks for InvalidOperationException.

Constructors

Expand table

InvalidOperationException()	Initializes a new instance of the InvalidOperationException class.
InvalidOperationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidOperationException class with serialized data.

InvalidOperationException(String)	Initializes a new instance of the InvalidOperationException class with a specified error message.
InvalidOperationException(String, Exception)	Initializes a new instance of the InvalidOperationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)

GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- IEnumerator
- ResourceSet
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InvalidOperationException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InvalidOperationException class.

Overloads

Expand table

InvalidOperationException()	Initializes a new instance of the InvalidOperationException class.
InvalidOperationException(String)	Initializes a new instance of the InvalidOperationException class with a specified error message.
InvalidOperationException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidOperationException class with serialized data.
InvalidOperationException(String, Exception)	Initializes a new instance of the InvalidOperationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

InvalidOperationException()

Initializes a new instance of the InvalidOperationException class.

C#

```
public InvalidOperationException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The requested operation cannot be performed." This message takes into account the current system culture.

The following table shows the initial property values for an instance of InvalidOperationException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidOperationException(String)

Initializes a new instance of the InvalidOperationException class with a specified error message.

C#

```
public InvalidOperationException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of InvalidOperationException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidOperationException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the InvalidOperationException class with serialized data.

```
C#

[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected InvalidOperationException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InvalidOperationException(String, Exception)

Initializes a new instance of the InvalidOperationException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public InvalidOperationException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of InvalidOperationException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InvalidProgramException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a program contains invalid Microsoft intermediate language (MSIL) or metadata. Generally this indicates a bug in the compiler that generated the program. This exception is also thrown when internal runtime implementation limits have been exceeded by the program.

C#

```
public sealed class InvalidProgramException : SystemException
```

Inheritance Object → Exception → SystemException → InvalidProgramException

Remarks

InvalidProgramException uses the HRESULT `COR_E_INVALIDPROGRAM`, which has the value 0x8013153A.

For a list of initial property values for an instance of InvalidProgramException, see the InvalidProgramException constructors.

Constructors

Expand table

InvalidProgramException()	Initializes a new instance of the InvalidProgramException class with default properties.
InvalidProgram Exception(String)	Initializes a new instance of the InvalidProgramException class with a specified error message.
InvalidProgram Exception(String, Exception)	Initializes a new instance of the InvalidProgramException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

InvalidProgramException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InvalidProgramException class.

Overloads

Expand table

InvalidProgramException()	Initializes a new instance of the InvalidProgramException class with default properties.
InvalidProgramException(String)	Initializes a new instance of the InvalidProgramException class with a specified error message.
InvalidProgramException(String, Exception)	Initializes a new instance of the InvalidProgramException class with a specified error message and a reference to the inner exception that is the cause of this exception.

InvalidProgramException()

Initializes a new instance of the InvalidProgramException class with default properties.

C#

```
public InvalidProgramException ();
```

Remarks

The constructor initializes the Exception.Message property of the new exception to a system-supplied message that describes the error, such as "Metadata is incorrect." This message takes into account the current system culture.

The following table shows the initial property values for an instance of InvalidProgramException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidProgramException(String)

Initializes a new instance of the InvalidProgramException class with a specified error message.

C#

```
public InvalidProgramException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

This constructor initializes the Exception.Message property of the new exception using the `message` parameter. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of InvalidProgramException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidProgramException(String, Exception)

Initializes a new instance of the InvalidProgramException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public InvalidProgramException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The `InnerException` property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the `InnerException` property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of InvalidProgramException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- InnerException
- Message
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InvalidTimeZoneException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when time zone information is invalid.

C#

```
public class InvalidTimeZoneException : Exception
```

Inheritance Object → Exception → InvalidTimeZoneException

Remarks

This exception is thrown when a TimeZoneInfo object contains invalid, incomplete, or missing data. It is thrown by the FindSystemTimeZoneById method when a time zone is found in the registry but contains corrupted data. It can also be thrown by the CreateCustomTimeZone method under any of the following conditions:

- The adjustment rules overlap.
- The adjustment rules contain one or more null elements.
- The sum of the Coordinated Universal Time (UTC) offset and daylight delta is out of range. (It is either greater than 14 hours or less than -14 hours.)
- A single date has multiple adjustment rules applied to it.

This exception is not thrown to indicate that the time zone is not defined on a particular system; the TimeZoneNotFoundException is used for this purpose.

Constructors

Expand table

InvalidTimeZoneException()	Initializes a new instance of the InvalidTimeZoneException class with a system-supplied message.
InvalidTimeZone Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidTimeZoneException class from serialized data.
InvalidTimeZoneException(String)	Initializes a new instance of the InvalidTimeZoneException class with the specified message string.
InvalidTimeZoneException(String, Exception)	Initializes a new instance of the InvalidTimeZoneException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

InvalidTimeZoneException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the InvalidTimeZoneException class.

Overloads

Expand table

InvalidTimeZoneException()	Initializes a new instance of the InvalidTimeZoneException class with a system-supplied message.
InvalidTimeZoneException(String)	Initializes a new instance of the InvalidTimeZoneException class with the specified message string.
InvalidTimeZoneException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the InvalidTimeZoneException class from serialized data.
InvalidTimeZoneException(String, Exception)	Initializes a new instance of the InvalidTimeZoneException class with a specified error message and a reference to the inner exception that is the cause of this exception.

InvalidTimeZoneException()

Initializes a new instance of the InvalidTimeZoneException class with a system-supplied message.

C#

```
public InvalidTimeZoneException ();
```

Remarks

This is the parameterless constructor of the InvalidTimeZoneException class. It initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Exception of type 'System.InvalidTimeZoneException' was thrown." This message is localized for the current system culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidTimeZoneException(String)

Initializes a new instance of the InvalidTimeZoneException class with the specified message string.

C#

```
public InvalidTimeZoneException (string? message);
```

Parameters

`message` String

A string that describes the exception.

Remarks

The string supplied as the `message` parameter is assigned to the Message property. It should be localized for the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

InvalidTimeZoneException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the InvalidTimeZoneException class from serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected InvalidTimeZoneException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that contains the serialized data.

`context` StreamingContext

The stream that contains the serialized data.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` parameter is `null`.

-or-

The `context` parameter is `null`.

Remarks

This constructor is not called directly by your code to instantiate the InvalidTimeZoneException object. Instead, it is called by the IFormatter object's Deserialize method when deserializing the InvalidTimeZoneException object from a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InvalidTimeZoneException(String, Exception)

Initializes a new instance of the InvalidTimeZoneException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public InvalidTimeZoneException (string? message, Exception?
innerException);
```

Parameters

`message` String

A string that describes the exception.

`innerException` Exception

The exception that is the cause of the current exception.

Examples

The following code tries to retrieve a TimeZoneInfo object that represents the Central Standard Time zone. If an InvalidTimeZoneException occurs in the `RetrieveTimeZone` method call, the exception handler wraps the exception in a new InvalidTimeZoneException object, which it returns to the caller. The caller's exception handler then displays information about both the outer and inner exceptions.

C#

```
private void HandleInnerException()
{
   string timeZoneName = "Any Standard Time";
   TimeZoneInfo tz;
   try
   {
      tz = RetrieveTimeZone(timeZoneName);
      Console.WriteLine("The time zone display name is {0}.",
tz.DisplayName);
   }
   catch (TimeZoneNotFoundException e)
   {
      Console.WriteLine("{0} thrown by application", e.GetType().Name);
      Console.WriteLine("   Message: {0}", e.Message);
      if (e.InnerException != null)
      {
         Console.WriteLine("   Inner Exception Information:");
         Exception innerEx = e.InnerException;
         while (innerEx != null)
         {
            Console.WriteLine("      {0}: {1}", innerEx.GetType().Name,
innerEx.Message);
            innerEx = innerEx.InnerException;
         }
      }
   }
}

private TimeZoneInfo RetrieveTimeZone(string tzName)
{
   try
   {
```

```
        return TimeZoneInfo.FindSystemTimeZoneById(tzName);
    }
    catch (TimeZoneNotFoundException ex1)
    {
        throw new TimeZoneNotFoundException(
              String.Format("The time zone '{0}' cannot be found.",
tzName),
              ex1);
    }
    catch (InvalidTimeZoneException ex2)
    {
        throw new InvalidTimeZoneException(
              String.Format("The time zone {0} contains invalid data.",
tzName),
              ex2);
    }
}
```

Remarks

Typically, you use this overload of the InvalidTimeZoneException class to handle an exception in a `try`...`catch` block. The `innerException` parameter should be a reference to the exception object handled in the `catch` block, or it can be `null`. This value is then assigned to the InvalidTimeZoneException object's InnerException property.

The `message` string is assigned to the Message property. The string should be localized for the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IObservable<T> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a provider for push-based notification.

C#

```
public interface IObservable<out T>
```

Type Parameters

T

The object that provides notification information.
This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Derived System.Diagnostics.DiagnosticListener
System.Web.Hosting.AspNetMemoryMonitor

Examples

The following example illustrates the observer design pattern. It defines a `Location` class that contains latitude and longitude information.

C#

```
public struct Location
{
   double lat, lon;

   public Location(double latitude, double longitude)
   {
      this.lat = latitude;
      this.lon = longitude;
   }
```

```
    public double Latitude
    { get { return this.lat; } }

    public double Longitude
    { get { return this.lon; } }
}
```

The `LocationTracker` class provides the IObservable<T> implementation. Its `TrackLocation` method is passed a nullable `Location` object that contains the latitude and longitude data. If the `Location` value is not `null`, the `TrackLocation` method calls the OnNext method of each observer.

C#

```
public class LocationTracker : IObservable<Location>
{
    public LocationTracker()
    {
        observers = new List<IObserver<Location>>();
    }

    private List<IObserver<Location>> observers;

    public IDisposable Subscribe(IObserver<Location> observer)
    {
        if (! observers.Contains(observer))
            observers.Add(observer);
        return new Unsubscriber(observers, observer);
    }

    private class Unsubscriber : IDisposable
    {
        private List<IObserver<Location>>_observers;
        private IObserver<Location> _observer;

        public Unsubscriber(List<IObserver<Location>> observers,
IObserver<Location> observer)
        {
            this._observers = observers;
            this._observer = observer;
        }

        public void Dispose()
        {
            if (_observer != null && _observers.Contains(_observer))
                _observers.Remove(_observer);
        }
    }

    public void TrackLocation(Nullable<Location> loc)
    {
        foreach (var observer in observers) {
```

```
            if (! loc.HasValue)
               observer.OnError(new LocationUnknownException());
            else
               observer.OnNext(loc.Value);
         }
      }

      public void EndTransmission()
      {
         foreach (var observer in observers.ToArray())
            if (observers.Contains(observer))
               observer.OnCompleted();

         observers.Clear();
      }
   }
```

If the `Location` value is `null`, the `TrackLocation` method instantiates a `LocationUnknownException` object, which is shown in the following example. It then calls each observer's OnError method and passes it the `LocationUnknownException` object. Note that `LocationUnknownException` derives from Exception, but does not add any new members.

C#

```
public class LocationUnknownException : Exception
{
   internal LocationUnknownException()
   { }
}
```

Observers register to receive notifications from a `LocationTracker` object by calling its IObservable<T>.Subscribe method, which assigns a reference to the observer object to a private generic List<T> object. The method returns an `Unsubscriber` object, which is an IDisposable implementation that enables observers to stop receiving notifications. The `LocationTracker` class also includes an `EndTransmission` method. When no further location data is available, the method calls each observer's OnCompleted method and then clears the internal list of observers.

In this example, the `LocationReporter` class provides the IObserver<T> implementation. It displays information about the current location to the console. Its constructor includes a `name` parameter, which enables the `LocationReporter` instance to identify itself in its string output. It also includes a `Subscribe` method, which wraps a call to the provider's Subscribe method. This allows the method to assign the returned IDisposable reference to a private variable. The `LocationReporter` class also includes an `Unsubscribe` method,

which calls the IDisposable.Dispose method of the object that is returned by the IObservable<T>.Subscribe method. The following code defines the `LocationReporter` class.

C#

```
using System;

public class LocationReporter : IObserver<Location>
{
   private IDisposable unsubscriber;
   private string instName;

   public LocationReporter(string name)
   {
      this.instName = name;
   }

   public string Name
   {  get{ return this.instName; } }

   public virtual void Subscribe(IObservable<Location> provider)
   {
      if (provider != null)
         unsubscriber = provider.Subscribe(this);
   }

   public virtual void OnCompleted()
   {
      Console.WriteLine("The Location Tracker has completed transmitting
data to {0}.", this.Name);
      this.Unsubscribe();
   }

   public virtual void OnError(Exception e)
   {
      Console.WriteLine("{0}: The location cannot be determined.",
this.Name);
   }

   public virtual void OnNext(Location value)
   {
      Console.WriteLine("{2}: The current location is {0}, {1}",
value.Latitude, value.Longitude, this.Name);
   }

   public virtual void Unsubscribe()
   {
      unsubscriber.Dispose();
   }
}
```

The following code then instantiates the provider and the observer.

```C#
using System;

class Program
{
   static void Main(string[] args)
   {
      // Define a provider and two observers.
      LocationTracker provider = new LocationTracker();
      LocationReporter reporter1 = new LocationReporter("FixedGPS");
      reporter1.Subscribe(provider);
      LocationReporter reporter2 = new LocationReporter("MobileGPS");
      reporter2.Subscribe(provider);

      provider.TrackLocation(new Location(47.6456, -122.1312));
      reporter1.Unsubscribe();
      provider.TrackLocation(new Location(47.6677, -122.1199));
      provider.TrackLocation(null);
      provider.EndTransmission();
   }
}
// The example displays output similar to the following:
//      FixedGPS: The current location is 47.6456, -122.1312
//      MobileGPS: The current location is 47.6456, -122.1312
//      MobileGPS: The current location is 47.6677, -122.1199
//      MobileGPS: The location cannot be determined.
//      The Location Tracker has completed transmitting data to MobileGPS.
```

Remarks

The IObserver<T> and IObservable<T> interfaces provide a generalized mechanism for push-based notification, also known as the observer design pattern. The IObservable<T> interface represents the class that sends notifications (the provider); the IObserver<T> interface represents the class that receives them (the observer). `T` represents the class that provides the notification information. In some push-based notifications, the IObserver<T> implementation and `T` can represent the same type.

The provider must implement a single method, Subscribe, that indicates that an observer wants to receive push-based notifications. Callers to the method pass an instance of the observer. The method returns an IDisposable implementation that enables observers to cancel notifications at any time before the provider has stopped sending them.

At any given time, a given provider may have zero, one, or multiple observers. The provider is responsible for storing references to observers and ensuring that they are valid before it sends notifications. The IObservable<T> interface does not make any assumptions about the number of observers or the order in which notifications are sent.

The provider sends the following three kinds of notifications to the observer by calling IObserver<T> methods:

- The current data. The provider can call the IObserver<T>.OnNext method to pass the observer a `T` object that has current data, changed data, or fresh data.
- An error condition. The provider can call the IObserver<T>.OnError method to notify the observer that some error condition has occurred.
- No further data. The provider can call the IObserver<T>.OnCompleted method to notify the observer that it has finished sending notifications.

Methods

Expand table

Subscribe(IObserver<T>)	Notifies the provider that an observer is to receive notifications.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IObserver<T>
- The Observer Design Pattern

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IObservable<T>.Subscribe(IObserver<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Notifies the provider that an observer is to receive notifications.

C#

```
public IDisposable Subscribe (IObserver<out T> observer);
```

Parameters

observer IObserver<T>

The object that is to receive notifications.

Returns

IDisposable

A reference to an interface that allows observers to stop receiving notifications before the provider has finished sending them.

Examples

The following example illustrates the Subscribe method for an application that reports latitude and longitude information. It defines an IList<T> collection object that stores references to all observers. It also returns a private class named `Unsubscriber` that implements the IDisposable interface and enables subscribers to stop receiving event notifications. See the Example section of the IObservable<T> topic for the complete example.

C#

```
private List<IObserver<Location>> observers;
```

```
public IDisposable Subscribe(IObserver<Location> observer)
{
   if (! observers.Contains(observer))
      observers.Add(observer);
   return new Unsubscriber(observers, observer);
}

private class Unsubscriber : IDisposable
{
   private List<IObserver<Location>>_observers;
   private IObserver<Location> _observer;

   public Unsubscriber(List<IObserver<Location>> observers,
IObserver<Location> observer)
   {
      this._observers = observers;
      this._observer = observer;
   }

   public void Dispose()
   {
      if (_observer != null && _observers.Contains(_observer))
         _observers.Remove(_observer);
   }
}
```

Remarks

The Subscribe method must be called to register an observer for push-based notifications. A typical implementation of the Subscribe method does the following:

- It stores a reference to the observer in a collection object, such as a List<T> object.
- It returns a reference to an IDisposable interface. This enables observers to unsubscribe (that is, to stop receiving notifications) before the provider has finished sending them and called the subscriber's OnCompleted method.

At any given time, a particular instance of an IObservable<T> implementation is responsible for handling all subscriptions and notifying all subscribers. Unless the documentation for a particular IObservable<T> implementation indicates otherwise, observers should make no assumptions about the IObservable<T> implementation, such as the order of notifications that multiple observers will receive.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IObserver<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IObserver<T> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a mechanism for receiving push-based notifications.

C#

```
public interface IObserver<in T>
```

Type Parameters

T

The object that provides notification information.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Derived Microsoft.Extensions.DiagnosticAdapter.DiagnosticSourceAdapter
System.Web.Hosting.LowPhysicalMemoryObserver
System.Web.Hosting.RecycleLimitObserver

Examples

The following example illustrates the observer design pattern. It defines a `Location` class that contains latitude and longitude information.

C#

```
public struct Location
{
   double lat, lon;

   public Location(double latitude, double longitude)
   {
      this.lat = latitude;
      this.lon = longitude;
   }
```

```
    public double Latitude
    { get { return this.lat; } }

    public double Longitude
    { get { return this.lon; } }
}
```

The `LocationReporter` class provides the IObserver<T> implementation. It displays information about the current location to the console. Its constructor includes a `name` parameter, which allows the `LocationReporter` instance to identify itself in its string output. It also includes a `Subscribe` method, which wraps a call to the provider's Subscribe method. This enables the method to assign the returned IDisposable reference to a private variable. The `LocationReporter` class also includes an `Unsubscribe` method, which calls the IDisposable.Dispose method of the object returned by the IObservable<T>.Subscribe method. The following code defines the `LocationReporter` class.

C#

```
using System;

public class LocationReporter : IObserver<Location>
{
    private IDisposable unsubscriber;
    private string instName;

    public LocationReporter(string name)
    {
        this.instName = name;
    }

    public string Name
    {  get{ return this.instName; } }

    public virtual void Subscribe(IObservable<Location> provider)
    {
        if (provider != null)
            unsubscriber = provider.Subscribe(this);
    }

    public virtual void OnCompleted()
    {
        Console.WriteLine("The Location Tracker has completed transmitting
data to {0}.", this.Name);
        this.Unsubscribe();
    }

    public virtual void OnError(Exception e)
    {
```

```
      Console.WriteLine("{0}: The location cannot be determined.",
this.Name);
   }

   public virtual void OnNext(Location value)
   {
      Console.WriteLine("{2}: The current location is {0}, {1}",
value.Latitude, value.Longitude, this.Name);
   }

   public virtual void Unsubscribe()
   {
      unsubscriber.Dispose();
   }
}
```

The `LocationTracker` class provides the IObservable<T> implementation. Its `TrackLocation` method is passed a nullable `Location` object that contains the latitude and longitude data. If the `Location` value is not `null`, the `TrackLocation` method calls the OnNext method of each observer.

C#

```
public class LocationTracker : IObservable<Location>
{
   public LocationTracker()
   {
      observers = new List<IObserver<Location>>();
   }

   private List<IObserver<Location>> observers;

   public IDisposable Subscribe(IObserver<Location> observer)
   {
      if (! observers.Contains(observer))
         observers.Add(observer);
      return new Unsubscriber(observers, observer);
   }

   private class Unsubscriber : IDisposable
   {
      private List<IObserver<Location>>_observers;
      private IObserver<Location> _observer;

      public Unsubscriber(List<IObserver<Location>> observers,
IObserver<Location> observer)
      {
         this._observers = observers;
         this._observer = observer;
      }

      public void Dispose()
```

```
        {
            if (_observer != null && _observers.Contains(_observer))
                _observers.Remove(_observer);
        }
    }

    public void TrackLocation(Nullable<Location> loc)
    {
        foreach (var observer in observers) {
            if (! loc.HasValue)
                observer.OnError(new LocationUnknownException());
            else
                observer.OnNext(loc.Value);
        }
    }

    public void EndTransmission()
    {
        foreach (var observer in observers.ToArray())
            if (observers.Contains(observer))
                observer.OnCompleted();

        observers.Clear();
    }
}
```

If the `Location` value is `null`, the `TrackLocation` method instantiates a `LocationNotFoundException` object, which is shown in the following example. It then calls each observer's OnError method and passes it the `LocationNotFoundException` object. Note that `LocationNotFoundException` derives from Exception but does not add any new members.

C#

```
public class LocationUnknownException : Exception
{
    internal LocationUnknownException()
    { }
}
```

Observers register to receive notifications from a `TrackLocation` object by calling its IObservable<T>.Subscribe method, which assigns a reference to the observer object to a private generic List<T> object. The method returns an `Unsubscriber` object, which is an IDisposable implementation that enables observers to stop receiving notifications. The `LocationTracker` class also includes an `EndTransmission` method. When no further location data is available, the method calls each observer's OnCompleted method and then clears the internal list of observers.

The following code then instantiates the provider and the observer.

C#

```
using System;

class Program
{
   static void Main(string[] args)
   {
      // Define a provider and two observers.
      LocationTracker provider = new LocationTracker();
      LocationReporter reporter1 = new LocationReporter("FixedGPS");
      reporter1.Subscribe(provider);
      LocationReporter reporter2 = new LocationReporter("MobileGPS");
      reporter2.Subscribe(provider);

      provider.TrackLocation(new Location(47.6456, -122.1312));
      reporter1.Unsubscribe();
      provider.TrackLocation(new Location(47.6677, -122.1199));
      provider.TrackLocation(null);
      provider.EndTransmission();
   }
}
// The example displays output similar to the following:
//      FixedGPS: The current location is 47.6456, -122.1312
//      MobileGPS: The current location is 47.6456, -122.1312
//      MobileGPS: The current location is 47.6677, -122.1199
//      MobileGPS: The location cannot be determined.
//      The Location Tracker has completed transmitting data to MobileGPS.
```

Remarks

The IObserver<T> and IObservable<T> interfaces provide a generalized mechanism for push-based notification, also known as the observer design pattern. The IObservable<T> interface represents the class that sends notifications (the provider); the IObserver<T> interface represents the class that receives them (the observer). `T` represents the class that provides the notification information.

An IObserver<T> implementation arranges to receive notifications from a provider (an IObservable<T> implementation) by passing an instance of itself to the provider's IObservable<T>.Subscribe method. This method returns an IDisposable object that can be used to unsubscribe the observer before the provider finishes sending notifications.

The IObserver<T> interface defines the following three methods that the observer must implement:

- The OnNext method, which is typically called by the provider to supply the observer with new data or state information.
- The OnError method, which is typically called by the provider to indicate that data is unavailable, inaccessible, or corrupted, or that the provider has experienced some other error condition.
- The OnCompleted method, which is typically called by the provider to indicate that it has finished sending notifications to observers.

Methods

Expand table

OnCompleted()	Notifies the observer that the provider has finished sending push-based notifications.
On Error(Exception)	Notifies the observer that the provider has experienced an error condition.
OnNext(T)	Provides the observer with new data.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IObservable<T>

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IObserver<T>.OnCompleted Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Notifies the observer that the provider has finished sending push-based notifications.

C#

```
public void OnCompleted ();
```

Examples

The following example provides an implementation of the OnCompleted method in a latitude/longitude tracking application. The method simply reports that no further data is available and calls the provider's IDisposable.Dispose implementation. See the Example section of the IObserver<T> topic for the complete example.

C#

```
public virtual void OnCompleted()
{
   Console.WriteLine("The Location Tracker has completed transmitting data
to {0}.", this.Name);
   this.Unsubscribe();
}
```

Remarks

When the observer's OnCompleted implementation is called, the OnCompleted method can optionally call the Dispose method of the IDisposable object that was returned to the observer when it called the IObservable<T>.Subscribe method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IObserver<T>.OnError(Exception) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Notifies the observer that the provider has experienced an error condition.

```
C#

public void OnError (Exception error);
```

Parameters

`error` Exception

An object that provides additional information about the error.

Examples

The following example provides an implementation of the OnError method in a latitude/longitude tracking application. The method simply reports that data is currently unavailable; it does not make use of the Exception object passed to it as a parameter. See the Example section of the IObserver<T> topic for the complete example.

```
C#

public virtual void OnError(Exception e)
{
   Console.WriteLine("{0}: The location cannot be determined.", this.Name);
}
```

Remarks

Although `error` is an object that is derived from System.Exception, it does not necessarily represent an exception that is thrown by the provider. It can also represent a routine or expected error condition, such as data that is missing or unavailable. The

OnError method should be seen as informational, and the provider should not expect the observer to provide error handling.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Observer Design Pattern Best Practices

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IObserver<T>.OnNext(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides the observer with new data.

```
C#

public void OnNext (T value);
```

Parameters

`value` T

The current notification information.

Examples

The following example provides an implementation of the OnNext method in a latitude/longitude tracking application. See the Example section of the IObserver<T> topic for the complete example.

```
C#

public virtual void OnNext(Location value)
{
   Console.WriteLine("{2}: The current location is {0}, {1}",
value.Latitude, value.Longitude, this.Name);
}
```

Remarks

After an observer has called a provider's IObservable<T>.Subscribe method, the provider calls the observer's OnNext method to provide notifications.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IParsable<TSelf> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a mechanism for parsing a string to a value.

C#

```
public interface IParsable<TSelf> where TSelf : IParsable<TSelf>
```

Type Parameters

`TSelf`

The type that implements this interface.

Derived System.Boolean
System.Byte
System.Char
System.DateOnly
System.DateTime
More...

Methods

Expand table

Parse(String, IFormatProvider)	Parses a string into a value.
TryParse(String, IFormatProvider, TSelf)	Tries to parse a string into a value.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IParsable<TSelf>.Parse(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a string into a value.

```
C#
public static abstract TSelf Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

TSelf

The result of parsing `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` is not representable by `TSelf`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IParsable<TSelf>.TryParse(String, IFormatProvider, TSelf) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to parse a string into a value.

```
C#

public static abstract bool TryParse (string? s, IFormatProvider? provider,
out TSelf result);
```

Parameters

s String

The string to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result TSelf

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IProgress<T> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a provider for progress updates.

C#

```
public interface IProgress<in T>
```

Type Parameters

T

The type of progress update value.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Derived System.Progress<T>

Methods

Expand table

Report(T)	Reports a progress update.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IProgress<T>.Report(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports a progress update.

C#

```
public void Report (T value);
```

Parameters

`value` T

The value of the updated progress.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IServiceProvider Interface

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.dll

Defines a mechanism for retrieving a service object; that is, an object that provides custom support to other objects.

C#

```
public interface IServiceProvider
```

Derived Microsoft.Extensions.DependencyInjection.IKeyedServiceProvider
Microsoft.Extensions.DependencyInjection.ServiceProvider
System.Activities.Presentation.ServiceManager
System.CommandLine.Binding.BindingContext
System.ComponentModel.DataAnnotations.ValidationContext
More...

Remarks

This interface is implemented by a class or value type that provides a service to other objects.

The GetService method of this interface obtains the object that provides the service.

The IServiceProvider interface is implemented by a number of types, including System.Web.HttpContext, System.ComponentModel.LicenseContext, System.ComponentModel.MarshalByValueComponent, and System.ComponentModel.Design.ServiceContainer.

Methods

Expand table

GetService(Type)	Gets the service object of the specified type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Type

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IServiceProvider.GetService(Type) Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.dll

Gets the service object of the specified type.

```
C#
public object? GetService (Type serviceType);
```

Parameters

`serviceType` Type

An object that specifies the type of service object to get.

Returns

Object

A service object of type `serviceType`.

-or-

`null` if there is no service object of type `serviceType`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ISpanFormattable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides functionality to format the string representation of an object into a span.

C#

```
public interface ISpanFormattable : IFormattable
```

Derived System.Byte
System.Char
System.DateOnly
System.DateTime
System.DateTimeOffset
More...

Implements IFormattable

Methods

Expand table

ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format. (Inherited from IFormattable)
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format, IFormatProvider? provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Remarks

- An implementation of this interface should produce the same string of characters as an implementation of ToString(String, IFormatProvider) on the same type.
- TryFormat should return false only if there is not enough space in the destination buffer. Any other failures should throw an exception.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ISpanParsable<TSelf> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a mechanism for parsing a span of characters to a value.

C#

```
public interface ISpanParsable<TSelf> : IParsable<TSelf> where TSelf :
ISpanParsable<TSelf>
```

Type Parameters

TSelf

The type that implements this interface.

Derived System.Boolean
System.Byte
System.Char
System.DateOnly
System.DateTime
More...

Implements IParsable<TSelf>

Methods

Expand table

Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, TSelf)	Tries to parse a span of characters into a value.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ISpanParsable<TSelf>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a span of characters into a value.

```
C#

public static abstract TSelf Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

TSelf

The result of parsing `s`.

Exceptions

FormatException

`s` is not in the correct format.

OverflowException

`s` is not representable by `TSelf`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ISpanParsable<TSelf>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to parse a span of characters into a value.

C#

```
public static abstract bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out TSelf result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result TSelf

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IUtf8SpanFormattable Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides functionality to format the string representation of an object into a span as UTF-8.

C#

```
public interface IUtf8SpanFormattable
```

Derived System.Byte
System.Char
System.DateOnly
System.DateTime
System.DateTimeOffset
More...

Methods

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IUtf8SpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format, IFormatProvider? provider);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Remarks

An implementation of this interface should produce the same string of characters as an implementation of ToString(String, IFormatProvider) or TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider) on the same type. TryFormat should return `false` only if there is not enough space in the destination buffer; any other failures should throw an exception.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IUtf8SpanParsable<TSelf> Interface

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a mechanism for parsing a span of UTF-8 characters to a value.

C#

```
public interface IUtf8SpanParsable<TSelf> where TSelf :
IUtf8SpanParsable<TSelf>
```

Type Parameters

`TSelf`

The type that implements this interface.

Derived System.Byte
System.Char
System.Decimal
System.Double
System.Half
More...

Methods

Expand table

Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, TSelf)	Tries to parse a span of UTF-8 characters into a value.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IUtf8SpanParsable<TSelf>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a span of UTF-8 characters into a value.

```
C#

public static abstract TSelf Parse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider);
```

Parameters

utf8Text ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

TSelf

The result of parsing `utf8Text`.

Exceptions

FormatException

`utf8Text` is not in the correct format.

OverflowException

`utf8Text` is not representable by `TSelf`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IUtf8SpanParsable<TSelf>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static abstract bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out TSelf result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` TSelf

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Lazy<T> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides support for lazy initialization.

```
C#
public class Lazy<T>
```

Type Parameters

T

The type of object that is being lazily initialized.

Inheritance Object → Lazy<T>

Derived System.Lazy<T,TMetadata>

Examples

The following example demonstrates the use of the Lazy<T> class to provide lazy initialization with access from multiple threads.

> **Note**
>
> The example uses the **Lazy<T>(Func<T>)** constructor. It also demonstrates the use of the **Lazy<T>(Func<T>, Boolean)** constructor (specifying `true` for `isThreadSafe`) and the **Lazy<T>(Func<T>, LazyThreadSafetyMode)** constructor (specifying **LazyThreadSafetyMode.ExecutionAndPublication** for `mode`). To switch to a different constructor, just change which constructors are commented out.
>
> For an example that demonstrates exception caching using the same constructors, see the **Lazy<T>(Func<T>)** constructor.

The example defines a `LargeObject` class that will be initialized lazily by one of several threads. The four key sections of code illustrate the creation of the initializer, the factory method, the actual initialization, and the constructor of the `LargeObject` class, which displays a message when the object is created. At the beginning of the `Main` method, the example creates the thread-safe lazy initializer for `LargeObject`:

C#

```
lazyLargeObject = new Lazy<LargeObject>(InitLargeObject);

// The following lines show how to use other constructors to achieve exactly
the
// same result as the previous line:
//lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, true);
//lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
//
LazyThreadSafetyMode.ExecutionAndPublication);
```

The factory method shows the creation of the object, with a placeholder for further initialization:

C#

```
static LargeObject InitLargeObject()
{
    LargeObject large = new
LargeObject(Thread.CurrentThread.ManagedThreadId);
    // Perform additional initialization here.
    return large;
}
```

Note that the first two code sections could be combined by using a lambda function, as shown here:

C#

```
lazyLargeObject = new Lazy<LargeObject>(() =>
{
    LargeObject large = new
LargeObject(Thread.CurrentThread.ManagedThreadId);
    // Perform additional initialization here.
    return large;
});
```

The example pauses, to indicate that an indeterminate period may elapse before lazy initialization occurs. When you press the **Enter** key, the example creates and starts three

threads. The `ThreadProc` method that's used by all three threads calls the Value property. The first time this happens, the `LargeObject` instance is created:

C#

```
LargeObject large = lazyLargeObject.Value;

// IMPORTANT: Lazy initialization is thread-safe, but it doesn't protect the
//            object after creation. You must lock the object before
accessing it,
//            unless the type is thread safe. (LargeObject is not thread
safe.)
lock(large)
{
    large.Data[0] = Thread.CurrentThread.ManagedThreadId;
    Console.WriteLine("Initialized by thread {0}; last used by thread {1}.",
        large.InitializedBy, large.Data[0]);
}
```

The constructor of the `LargeObject` class, which includes the last key section of code, displays a message and records the identity of the initializing thread. The output from the program appears at the end of the full code listing.

C#

```
int initBy = 0;
public LargeObject(int initializedBy)
{
    initBy = initializedBy;
    Console.WriteLine("LargeObject was created on thread id {0}.", initBy);
}
```

> **Note**
>
> For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;
```

```
    static LargeObject InitLargeObject()
    {
        LargeObject large = new
LargeObject(Thread.CurrentThread.ManagedThreadId);
        // Perform additional initialization here.
        return large;
    }

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not created
until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>(InitLargeObject);

        // The following lines show how to use other constructors to achieve
exactly the
        // same result as the previous line:
        //lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, true);
        //lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
        //
LazyThreadSafetyMode.ExecutionAndPublication);

        Console.WriteLine(
            "\r\nLargeObject is not created until you access the Value
property of the lazy" +
            "\r\ninitializer. Press Enter to create LargeObject.");
        Console.ReadLine();

        // Create and start 3 threads, each of which uses LargeObject.
        Thread[] threads = new Thread[3];
        for (int i = 0; i < 3; i++)
        {
            threads[i] = new Thread(ThreadProc);
            threads[i].Start();
        }

        // Wait for all 3 threads to finish.
        foreach (Thread t in threads)
        {
            t.Join();
        }

        Console.WriteLine("\r\nPress Enter to end the program");
        Console.ReadLine();
    }

    static void ThreadProc(object state)
    {
        LargeObject large = lazyLargeObject.Value;

        // IMPORTANT: Lazy initialization is thread-safe, but it doesn't
protect the
        //            object after creation. You must lock the object before
accessing it,
```

```
        //            unless the type is thread safe. (LargeObject is not
thread safe.)
        lock(large)
        {
            large.Data[0] = Thread.CurrentThread.ManagedThreadId;
            Console.WriteLine("Initialized by thread {0}; last used by
thread {1}.",
                large.InitializedBy, large.Data[0]);
        }
    }
}

class LargeObject
{
    public int InitializedBy { get { return initBy; } }

    int initBy = 0;
    public LargeObject(int initializedBy)
    {
        initBy = initializedBy;
        Console.WriteLine("LargeObject was created on thread id {0}.",
initBy);
    }

    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

LargeObject is not created until you access the Value property of the lazy
initializer. Press Enter to create LargeObject.

LargeObject was created on thread id 3.
Initialized by thread 3; last used by thread 3.
Initialized by thread 3; last used by thread 4.
Initialized by thread 3; last used by thread 5.

Press Enter to end the program
 */
```

Remarks

Use lazy initialization to defer the creation of a large or resource-intensive object, or the execution of a resource-intensive task, particularly when such creation or execution might not occur during the lifetime of the program.

To prepare for lazy initialization, you create an instance of Lazy<T>. The type argument of the Lazy<T> object that you create specifies the type of the object that you want to initialize lazily. The constructor that you use to create the Lazy<T> object determines

the characteristics of the initialization. Lazy initialization occurs the first time the Lazy<T>.Value property is accessed.

In most cases, choosing a constructor depends on your answers to two questions:

- Will the lazily initialized object be accessed from more than one thread? If so, the Lazy<T> object might create it on any thread. You can use one of the simple constructors whose default behavior is to create a thread-safe Lazy<T> object, so that only one instance of the lazily instantiated object is created no matter how many threads try to access it. To create a Lazy<T> object that is not thread safe, you must use a constructor that enables you to specify no thread safety.

  > ⊗ **Caution**
  >
  > Making the **Lazy<T>** object thread safe does not protect the lazily initialized object. If multiple threads can access the lazily initialized object, you must make its properties and methods safe for multithreaded access.

- Does lazy initialization require a lot of code, or does the lazily initialized object have a parameterless constructor that does everything you need and doesn't throw exceptions? If you need to write initialization code or if exceptions need to be handled, use one of the constructors that takes a factory method. Write your initialization code in the factory method.

The following table shows which constructor to choose, based on these two factors:

Expand table

Object will be accessed by	If no initialization code is required (parameterless constructor), use	If initialization code is required, use
Multiple threads	Lazy<T>()	Lazy<T>(Func<T>)
One thread	Lazy<T>(Boolean) with `isThreadSafe` set to `false`.	Lazy<T>(Func<T>, Boolean) with `isThreadSafe` set to `false`.

You can use a lambda expression to specify the factory method. This keeps all the initialization code in one place. The lambda expression captures the context, including any arguments you pass to the lazily initialized object's constructor.

Exception caching When you use factory methods, exceptions are cached. That is, if the factory method throws an exception the first time a thread tries to access the Value property of the Lazy<T> object, the same exception is thrown on every subsequent

attempt. This ensures that every call to the Value property produces the same result and avoids subtle errors that might arise if different threads get different results. The Lazy<T> stands in for an actual `T` that otherwise would have been initialized at some earlier point, usually during startup. A failure at that earlier point is usually fatal. If there is a potential for a recoverable failure, we recommend that you build the retry logic into the initialization routine (in this case, the factory method), just as you would if you weren't using lazy initialization.

Alternative to locking In certain situations, you might want to avoid the overhead of the Lazy<T> object's default locking behavior. In rare situations, there might be a potential for deadlocks. In such cases, you can use the Lazy<T>(LazyThreadSafetyMode) or Lazy<T>(Func<T>, LazyThreadSafetyMode) constructor, and specify LazyThreadSafetyMode.PublicationOnly. This enables the Lazy<T> object to create a copy of the lazily initialized object on each of several threads if the threads call the Value property simultaneously. The Lazy<T> object ensures that all threads use the same instance of the lazily initialized object and discards the instances that are not used. Thus, the cost of reducing the locking overhead is that your program might sometimes create and discard extra copies of an expensive object. In most cases, this is unlikely. The examples for the Lazy<T>(LazyThreadSafetyMode) and Lazy<T>(Func<T>, LazyThreadSafetyMode) constructors demonstrate this behavior.

> **Important**
>
> When you specify **LazyThreadSafetyMode.PublicationOnly**, exceptions are never cached, even if you specify a factory method.

Equivalent constructors In addition to enabling the use of LazyThreadSafetyMode.PublicationOnly, the Lazy<T>(LazyThreadSafetyMode) and Lazy<T>(Func<T>, LazyThreadSafetyMode) constructors can duplicate the functionality of the other constructors. The following table shows the parameter values that produce equivalent behavior.

Expand table

To create a **Lazy<T>** object that is	For constructors that have a `LazyThreadSafetyMode` `mode` parameter, set `mode` to	For constructors that have a Boolean `isThreadSafe` parameter, set `isThreadSafe` to	For constructors with no thread safety parameters
Fully thread safe; uses locking to	ExecutionAndPublication	`true`	All such constructors are

To create a Lazy<T> object that is	For constructors that have a `LazyThreadSafetyMode` `mode` parameter, set `mode` to	For constructors that have a Boolean `isThreadSafe` parameter, set `isThreadSafe` to	For constructors with no thread safety parameters
ensure that only one thread initializes the value.			fully thread safe.
Not thread safe.	None	`false`	Not applicable.
Fully thread safe; threads race to initialize the value.	PublicationOnly	Not applicable.	Not applicable.

Other capabilities For information about the use of Lazy<T> with thread-static fields, or as the backing store for properties, see Lazy Initialization.

Constructors

Expand table

Lazy<T>()	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type is used.
Lazy<T>(Boolean)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type and the specified initialization mode are used.
Lazy<T>(Func<T>)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function is used.
Lazy<T>(Func<T>, Boolean)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function and initialization mode are used.
Lazy<T>(Func<T>, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T> class that uses the specified initialization function and thread-safety mode.
Lazy<T>(LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T> class that uses the parameterless constructor of `T` and the specified thread-safety mode.
Lazy<T>(T)	Initializes a new instance of the Lazy<T> class that uses a preinitialized specified value.

Properties

Expand table

IsValueCreated	Gets a value that indicates whether a value has been created for this Lazy<T> instance.
Value	Gets the lazily initialized value of the current Lazy<T> instance.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the Value property for this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

By default, all public and protected members of the Lazy<T> class are thread safe and may be used concurrently from multiple threads. These thread-safety guarantees may

be removed optionally and per instance, using parameters to the type's constructors.

See also

- LazyThreadSafetyMode
- Lazy Initialization
- Lazy Expressions (F#)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lazy<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Lazy<T> class.

Overloads

Expand table

Lazy<T>()	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type is used.
Lazy<T>(Boolean)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type and the specified initialization mode are used.
Lazy<T>(Func<T>)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function is used.
Lazy<T>(LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T> class that uses the parameterless constructor of `T` and the specified thread-safety mode.
Lazy<T>(T)	Initializes a new instance of the Lazy<T> class that uses a preinitialized specified value.
Lazy<T>(Func<T>, Boolean)	Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function and initialization mode are used.
Lazy<T>(Func<T>, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T> class that uses the specified initialization function and thread-safety mode.

Lazy<T>()

Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type is used.

```C#
public Lazy ();
```

Examples

The following example demonstrates the use of this constructor. It also illustrates the use of the Lazy<T>(Boolean) constructor (specifying `true` for `isThreadSafe`) and the Lazy<T>(LazyThreadSafetyMode) constructor (specifying LazyThreadSafetyMode.ExecutionAndPublication for `mode`). To switch to a different constructor, just change which constructors are commented out.

The example defines a `LargeObject` class that will be initialized lazily by one of several threads. The two key lines of code in this example are the creation of the initializer and the actual initialization. At the beginning of the `Main` method, the example creates the thread-safe lazy initializer for `LargeObject`:

```C#
lazyLargeObject = new Lazy<LargeObject>();

// The following lines show how to use other constructors to achieve
exactly the
// same result as the previous line:
//lazyLargeObject = new Lazy<LargeObject>(true);
//lazyLargeObject = new Lazy<LargeObject>
(LazyThreadSafetyMode.ExecutionAndPublication);
```

The example creates and starts three threads that block on a ManualResetEvent object, so that the example can release the threads all at once. The `ThreadProc` method that's used by all three threads calls the Value property to get the `LargeObject` instance:

```C#
LargeObject large = lazyLargeObject.Value;
```

The Lazy<T> class provides locking, so that only one thread is allowed to create the `LargeObject` instance. The example demonstrates that the other threads all get the same instance.

ⓘ **Note**

For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>();

        // The following lines show how to use other constructors to
achieve exactly the
        // same result as the previous line:
        //lazyLargeObject = new Lazy<LargeObject>(true);
        //lazyLargeObject = new Lazy<LargeObject>
(LazyThreadSafetyMode.ExecutionAndPublication);

        Console.WriteLine(
            "\r\nLargeObject is not created until you access the Value
property of the lazy" +
            "\r\ninitializer. Press Enter to create LargeObject.");
        Console.ReadLine();

        // Create and start 3 threads, passing the same blocking event to
all of them.
        ManualResetEvent startingGate = new ManualResetEvent(false);
        Thread[] threads = { new Thread(ThreadProc), new
Thread(ThreadProc), new Thread(ThreadProc) };
        foreach (Thread t in threads)
        {
            t.Start(startingGate);
        }

        // Give all 3 threads time to start and wait, then release them
all at once.
        Thread.Sleep(100);
        startingGate.Set();

        // Wait for all 3 threads to finish. (The order doesn't matter.)
        foreach (Thread t in threads)
        {
            t.Join();
```

```
        }

        Console.WriteLine("\r\nPress Enter to end the program");
        Console.ReadLine();
    }

    static void ThreadProc(object state)
    {
        // Wait for the signal.
        ManualResetEvent waitForStart = (ManualResetEvent) state;
        waitForStart.WaitOne();

        LargeObject large = lazyLargeObject.Value;

        // The following line introduces an artificial delay to
exaggerate the race condition.
        Thread.Sleep(5);

        // IMPORTANT: Lazy initialization is thread-safe, but it doesn't
protect the
        //            object after creation. You must lock the object
before accessing it,
        //            unless the type is thread safe. (LargeObject is not
thread safe.)
        lock(large)
        {
            large.Data[0] = Thread.CurrentThread.ManagedThreadId;
            Console.WriteLine("Initialized by thread {0}; last used by
thread {1}.",
                large.InitializedBy, large.Data[0]);
        }
    }
}

class LargeObject
{
    int initBy = 0;
    public int InitializedBy { get { return initBy; } }

    public LargeObject()
    {
        initBy = Thread.CurrentThread.ManagedThreadId;
        Console.WriteLine("LargeObject was created on thread id {0}.",
initBy);
    }
    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

LargeObject is not created until you access the Value property of the
lazy
initializer. Press Enter to create LargeObject.

LargeObject was created on thread id 4.
```

```
 Initialized by thread 4; last used by thread 3.
 Initialized by thread 4; last used by thread 4.
 Initialized by thread 4; last used by thread 5.

 Press Enter to end the program
  */
```

Remarks

An instance that is created with this constructor may be used concurrently from multiple threads.

The thread safety mode of a Lazy<T> instance that is initialized with this constructor is LazyThreadSafetyMode.ExecutionAndPublication. The thread safety mode describes the behavior when multiple threads try to initialize the Lazy<T> instance.

A Lazy<T> instance that is created with this constructor does not cache exceptions. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T>(Boolean)

Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the parameterless constructor of the target type and the specified initialization mode are used.

```
C#

public Lazy (bool isThreadSafe);
```

Parameters

`isThreadSafe` Boolean

`true` to make this instance usable concurrently by multiple threads; `false` to make the instance usable by only one thread at a time.

Examples

The following example demonstrates the use of this constructor to create a lazy initializer that is not thread safe, for scenarios where all access to the lazily initialized object occurs on the same thread. It also demonstrates the use of the Lazy<T> (LazyThreadSafetyMode) constructor (specifying LazyThreadSafetyMode.None for `mode`. To switch to a different constructor, just change which constructor is commented out.

> ⓘ **Note**
>
> For code that demonstrates how to use this constructor in multithreaded scenarios (specifying `true` for `isThreadSafe`), see the example for the **Lazy<T>()** constructor.

The example defines a `LargeObject` class that will be initialized lazily. In the `Main` method, the example creates a Lazy<T> instance and then pauses. When you press the **Enter** key, the example accesses the Value property of the Lazy<T> instance, which causes initialization to occur. The constructor of the `LargeObject` class displays a console message.

> ⓘ **Note**

For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>(false);

        // The following lines show how to use other constructors to
achieve exactly the
        // same result as the previous line:
        //lazyLargeObject = new Lazy<LargeObject>
(LazyThreadSafetyMode.None);

        Console.WriteLine(
            "\r\nLargeObject is not created until you access the Value
property of the lazy" +
            "\r\ninitializer. Press Enter to create LargeObject.");
        Console.ReadLine();

        LargeObject large = lazyLargeObject.Value;

        large.Data[11] = 89;

        Console.WriteLine("\r\nPress Enter to end the program");
        Console.ReadLine();
    }
}

class LargeObject
{
    public LargeObject()
    {
        Console.WriteLine("LargeObject was created on thread id {0}.",
            Thread.CurrentThread.ManagedThreadId);
    }
    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:
```

```
LargeObject is not created until you access the Value property of the
lazy
initializer. Press Enter to create LargeObject.

LargeObject was created on thread id 1.

Press Enter to end the program
 */
```

Remarks

The thread safety mode of a Lazy<T> instance that is initialized with this constructor is LazyThreadSafetyMode.ExecutionAndPublication if `isThreadSafe` is `true`; otherwise, the mode is LazyThreadSafetyMode.None. The thread safety mode describes the behavior when multiple threads try to initialize the Lazy<T> instance. To specify the LazyThreadSafetyMode.PublicationOnly mode, use the Lazy<T>(Func<T>, LazyThreadSafetyMode) or Lazy<T>(LazyThreadSafetyMode) constructor.

A Lazy<T> instance that is created with this constructor does not cache exceptions. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T>(Func<T>)

Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function is used.

```
C#

public Lazy (Func<T> valueFactory);
```

Parameters

`valueFactory` Func<T>

The delegate that is invoked to produce the lazily initialized value when it is needed.

Exceptions

ArgumentNullException

`valueFactory` is `null`.

Examples

The following example demonstrates the use of this constructor to provide lazy initialization with exception caching. It also demonstrates the use of the Lazy<T> (Func<T>, Boolean) constructor (specifying `true` for `isThreadSafe`) and the Lazy<T> (Func<T>, LazyThreadSafetyMode) constructor (specifying LazyThreadSafetyMode.ExecutionAndPublication for `mode`). To switch to a different constructor, just change which constructors are commented out.

The example defines a `LargeObject` class that will be initialized lazily by one of several threads. The three key sections of code illustrate the creation of the initializer, the actual initialization, and the constructor of the `LargeObject` class, which demonstrates exception caching. At the beginning of the `Main` method, the example creates the thread-safe lazy initializer for `LargeObject`:

```
C#

lazyLargeObject = new Lazy<LargeObject>(InitLargeObject);

// The following lines show how to use other constructors to achieve
exactly the
// same result as the previous line:
//lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, true);
//lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
LazyThreadSafetyMode.ExecutionAndPublication);
```

The example creates and starts three threads. The `ThreadProc` method that's used by all three threads calls the Value property to get the `LargeObject` instance:

```C#
try
{
    LargeObject large = lazyLargeObject.Value;

    // IMPORTANT: Lazy initialization is thread-safe, but it doesn't protect the
    //            object after creation. You must lock the object before accessing it,
    //            unless the type is thread safe. (LargeObject is not thread safe.)
    lock(large)
    {
        large.Data[0] = Thread.CurrentThread.ManagedThreadId;
        Console.WriteLine("Initialized by thread {0}; last used by thread {1}.",
            large.InitializedBy, large.Data[0]);
    }
}
catch (ApplicationException aex)
{
    Console.WriteLine("Exception: {0}", aex.Message);
}
```

In the constructor of the `LargeObject` class, the third key section of code throws an exception the first time a `LargeObject` instance is created, but thereafter allows instance creation to occur:

```C#
static int instanceCount = 0;
public LargeObject()
{
    if (1 == Interlocked.Increment(ref instanceCount))
    {
        throw new ApplicationException("Throw only ONCE.");
    }

    initBy = Thread.CurrentThread.ManagedThreadId;
    Console.WriteLine("LargeObject was created on thread id {0}.", initBy);
}
```

When the example is run, the first thread that tries to create an instance of `LargeObject` fails, and the exception is caught. You might expect that the next thread

would successfully create an instance, but the Lazy<T> object has cached the exception. Because of this, all three threads throw the exception.

> **Note**
>
> For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    static LargeObject InitLargeObject()
    {
        return new LargeObject();
    }

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>(InitLargeObject);

        // The following lines show how to use other constructors to
achieve exactly the
        // same result as the previous line:
        //lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, true);
        //lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
LazyThreadSafetyMode.ExecutionAndPublication);

        Console.WriteLine(
            "\r\nLargeObject is not created until you access the Value
property of the lazy" +
            "\r\ninitializer. Press Enter to create LargeObject.");
        Console.ReadLine();

        // Create and start 3 threads, each of which tries to use
LargeObject.
        Thread[] threads = { new Thread(ThreadProc), new
Thread(ThreadProc), new Thread(ThreadProc) };
        foreach (Thread t in threads)
        {
            t.Start();
        }
```

```
        // Wait for all 3 threads to finish. (The order doesn't matter.)
        foreach (Thread t in threads)
        {
            t.Join();
        }

        Console.WriteLine("\r\nPress Enter to end the program");
        Console.ReadLine();
    }

    static void ThreadProc(object state)
    {
        try
        {
            LargeObject large = lazyLargeObject.Value;

            // IMPORTANT: Lazy initialization is thread-safe, but it
doesn't protect the
            //            object after creation. You must lock the object
before accessing it,
            //            unless the type is thread safe. (LargeObject is
not thread safe.)
            lock(large)
            {
                large.Data[0] = Thread.CurrentThread.ManagedThreadId;
                Console.WriteLine("Initialized by thread {0}; last used
by thread {1}.",
                    large.InitializedBy, large.Data[0]);
            }
        }
        catch (ApplicationException aex)
        {
            Console.WriteLine("Exception: {0}", aex.Message);
        }
    }
}

class LargeObject
{
    int initBy = 0;
    public int InitializedBy { get { return initBy; } }

    static int instanceCount = 0;
    public LargeObject()
    {
        if (1 == Interlocked.Increment(ref instanceCount))
        {
            throw new ApplicationException("Throw only ONCE.");
        }

        initBy = Thread.CurrentThread.ManagedThreadId;
        Console.WriteLine("LargeObject was created on thread id {0}.",
initBy);
    }
```

```
    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

LargeObject is not created until you access the Value property of the
lazy
initializer. Press Enter to create LargeObject.

Exception: Throw only ONCE.
Exception: Throw only ONCE.
Exception: Throw only ONCE.

Press Enter to end the program
 */
```

Remarks

An instance that is created with this constructor may be used concurrently from multiple threads.

The thread safety mode of a Lazy<T> instance that is initialized with this constructor is LazyThreadSafetyMode.ExecutionAndPublication. The thread safety mode describes the behavior when multiple threads try to initialize the Lazy<T> instance.

Exceptions that are thrown by `valueFactory` are cached. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Lazy<T>(LazyThreadSafetyMode)

Initializes a new instance of the Lazy<T> class that uses the parameterless constructor of `T` and the specified thread-safety mode.

```
C#
public Lazy (System.Threading.LazyThreadSafetyMode mode);
```

Parameters

`mode` LazyThreadSafetyMode

One of the enumeration values that specifies the thread safety mode.

Exceptions

ArgumentOutOfRangeException

`mode` contains an invalid value.

Examples

The following example demonstrates the use of this constructor to create a lazy initializer that enables multiple threads to race to create an object lazily. Multiple threads might succeed in creating instances, but all threads use the instance that was created first.

> ⓘ **Note**
>
> For an example that demonstrates how to use this constructor in single-threaded scenarios (specifying **LazyThreadSafetyMode.None** for `mode`), see the **Lazy<T>(Boolean)** constructor. For an example that demonstrates how to use this constructor to provide locking instead of race conditions in multithreaded scenarios (specifying **LazyThreadSafetyMode.ExecutionAndPublication** for `mode`), see the **Lazy<T>()** constructor.

The example defines a `LargeObject` class that will be initialized lazily by any of several threads. The three key sections of code illustrate the creation of the initializer, the actual initialization, and the constructor and finalizer of the `LargeObject` class. At the beginning of the `Main` method, the example creates the Lazy<T> object that performs lazy initialization of the `LargeObject`:

```
C#

lazyLargeObject = new Lazy<LargeObject>
(LazyThreadSafetyMode.PublicationOnly);
```

The example creates and starts three threads that block on a ManualResetEvent object, so that the example can release the threads all at once. In the `ThreadProc` method that's used by all three threads, calling the Value property creates the `LargeObject` instance:

```
C#

LargeObject large = lazyLargeObject.Value;
```

Because the constructor for the Lazy<T> instance specified LazyThreadSafetyMode.PublicationOnly, all three threads are allowed to create `LargeObject` instances. The example demonstrates this by displaying console messages in the constructor and in the finalizer of the `LargeObject` class:

```
C#

public LargeObject()
{
    initBy = Thread.CurrentThread.ManagedThreadId;
    Console.WriteLine("Constructor: Instance initializing on thread {0}",
initBy);
}

~LargeObject()
{
    Console.WriteLine("Finalizer: Instance was initialized on {0}",
initBy);
}
```

However, the Lazy<T> object ensures that only one instance is used by all threads. The output from the example shows that all three threads use the same instance, and also shows that the other two instances can be reclaimed by garbage collection.

> **Note**
>
> For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>
(LazyThreadSafetyMode.PublicationOnly);

        // Create and start 3 threads, passing the same blocking event to
all of them.
        ManualResetEvent startingGate = new ManualResetEvent(false);
        Thread[] threads = { new Thread(ThreadProc), new
Thread(ThreadProc), new Thread(ThreadProc) };
        foreach (Thread t in threads)
        {
            t.Start(startingGate);
        }

        // Give all 3 threads time to start and wait, then release them
all at once.
        Thread.Sleep(50);
        startingGate.Set();

        // Wait for all 3 threads to finish. (The order doesn't matter.)
        foreach (Thread t in threads)
        {
            t.Join();
        }

        Console.WriteLine(
            "\r\nThreads are complete. Running GC.Collect() to reclaim
the extra instances.");

        GC.Collect();

        // Allow time for garbage collection, which happens
```

```
asynchronously.
        Thread.Sleep(100);

        Console.WriteLine(
            "\r\nNote that all three threads used the instance that was
not collected.");
        Console.WriteLine("Press Enter to end the program");
        Console.ReadLine();
    }

    static void ThreadProc(object state)
    {
        // Wait for the signal.
        ManualResetEvent waitForStart = (ManualResetEvent) state;
        waitForStart.WaitOne();

        LargeObject large = lazyLargeObject.Value;

        // The following line introduces an artificial delay, to
exaggerate the race
        // condition.
        Thread.Sleep(5);

        // IMPORTANT: Lazy initialization is thread-safe, but it doesn't
protect the
        //            object after creation. You must lock the object
before accessing it,
        //            unless the type is thread safe. (LargeObject is not
thread safe.)
        lock(large)
        {
            large.Data[0] = Thread.CurrentThread.ManagedThreadId;
            Console.WriteLine("LargeObject was initialized by thread {0};
last used by thread {1}.",
                large.InitializedBy, large.Data[0]);
        }
    }
}

class LargeObject
{
    int initBy = -1;
    public int InitializedBy { get { return initBy; } }

    public LargeObject()
    {
        initBy = Thread.CurrentThread.ManagedThreadId;
        Console.WriteLine("Constructor: Instance initializing on thread
{0}", initBy);
    }

    ~LargeObject()
    {
        Console.WriteLine("Finalizer: Instance was initialized on {0}",
initBy);
```

```
    }

    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

Constructor: Instance initializing on thread 4
Constructor: Instance initializing on thread 3
Constructor: Instance initializing on thread 5
LargeObject was initialized by thread 4; last used by thread 4.
LargeObject was initialized by thread 4; last used by thread 5.
LargeObject was initialized by thread 4; last used by thread 3.

Threads are complete. Running GC.Collect() to reclaim the extra
instances.
Finalizer: Instance was initialized on 3
Finalizer: Instance was initialized on 5

Note that all three threads used the instance that was not collected.
Press Enter to end the program

Instance finalizing; initialized on 4
 */
```

Remarks

The thread safety mode of a Lazy<T> instance describes the behavior when multiple threads try to initialize the Lazy<T> instance.

A Lazy<T> instance that is created with this constructor does not cache exceptions. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- LazyThreadSafetyMode
- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T>(T)

Initializes a new instance of the Lazy<T> class that uses a preinitialized specified value.

```
C#

public Lazy (T value);
```

Parameters

`value` T

The preinitialized value to be used.

Remarks

An instance created with this constructor is usable by multiple threads concurrently.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Lazy<T>(Func<T>, Boolean)

Initializes a new instance of the Lazy<T> class. When lazy initialization occurs, the specified initialization function and initialization mode are used.

```
C#
```

```
public Lazy (Func<T> valueFactory, bool isThreadSafe);
```

Parameters

`valueFactory` Func<T>

The delegate that is invoked to produce the lazily initialized value when it is needed.

`isThreadSafe` Boolean

`true` to make this instance usable concurrently by multiple threads; `false` to make this instance usable by only one thread at a time.

Exceptions

ArgumentNullException

`valueFactory` is `null`.

Examples

The following example demonstrates the use of this constructor to provide lazy initialization with exception caching, in a scenario with a single thread. It also demonstrates the use of the Lazy<T> constructor (specifying LazyThreadSafetyMode.None for `mode`). To switch to that constructor, just change which constructor is commented out.

> **Note**
>
> For code that demonstrates how to use this constructor in multithreaded scenarios (specifying `true` for `isThreadSafe`), see the example for the **Lazy<T> (Func<T>)** constructor.

The example defines a `LargeObject` class that will be initialized lazily by one of several threads. The three key sections of code illustrate the creation of the initializer, the actual initialization, and the constructor of the `LargeObject` class, which demonstrates exception caching. At the beginning of the `Main` method, the example creates the thread-safe lazy initializer for `LargeObject`:

```
C#
```

```
lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, false);

// The following lines show how to use other constructors to achieve
exactly the
// same result as the previous line:
//lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
LazyThreadSafetyMode.None);
```

In the call to the constructor, the `isThreadSafe` parameter is `false`, so the Lazy<T> is not thread safe. Because it's not thread safe, the example calls the Value property three times on the same thread:

C#

```
for (int i = 0; i < 3; i++)
{
    try
    {
        LargeObject large = lazyLargeObject.Value;
        large.Data[11] = 89;
    }
    catch (ApplicationException aex)
    {
        Console.WriteLine("Exception: {0}", aex.Message);
    }
}
```

In the constructor of the `LargeObject` class, the third key section of code throws an exception the first time a `LargeObject` instance is created, but thereafter allows instance creation to occur:

C#

```
static bool pleaseThrow = true;
public LargeObject()
{
    if (pleaseThrow)
    {
        pleaseThrow = false;
        throw new ApplicationException("Throw only ONCE.");
    }

    Console.WriteLine("LargeObject was created on thread id {0}.",
        Thread.CurrentThread.ManagedThreadId);
}
```

When the example is run, the first attempt to create an instance of `LargeObject` fails, and the exception is caught. You might expect that the next attempt would succeed, but the Lazy<T> object has cached the exception. Because of this, all three attempts throw the exception.

> **Note**
>
> For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    static LargeObject InitLargeObject()
    {
        return new LargeObject();
    }

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>(InitLargeObject, false);

        // The following lines show how to use other constructors to
achieve exactly the
        // same result as the previous line:
        //lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
LazyThreadSafetyMode.None);

        Console.WriteLine(
            "\r\nLargeObject is not created until you access the Value
property of the lazy" +
            "\r\ninitializer. Press Enter to create LargeObject (three
tries).");
        Console.ReadLine();

        for (int i = 0; i < 3; i++)
        {
            try
            {
                LargeObject large = lazyLargeObject.Value;
```

```
                large.Data[11] = 89;
            }
            catch (ApplicationException aex)
            {
                Console.WriteLine("Exception: {0}", aex.Message);
            }
        }

        Console.WriteLine("\r\nPress Enter to end the program");
        Console.ReadLine();
    }
}

class LargeObject
{
    static bool pleaseThrow = true;
    public LargeObject()
    {
        if (pleaseThrow)
        {
            pleaseThrow = false;
            throw new ApplicationException("Throw only ONCE.");
        }

        Console.WriteLine("LargeObject was created on thread id {0}.",
            Thread.CurrentThread.ManagedThreadId);
    }
    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

LargeObject is not created until you access the Value property of the
lazy
initializer. Press Enter to create LargeObject (three tries).

Exception: Throw only ONCE.
Exception: Throw only ONCE.
Exception: Throw only ONCE.

Press Enter to end the program
 */
```

Remarks

The thread safety mode of a Lazy<T> instance that is initialized with this constructor is LazyThreadSafetyMode.ExecutionAndPublication if `isThreadSafe` is `true`; otherwise, the mode is LazyThreadSafetyMode.None. The thread safety mode describes the behavior when multiple threads try to initialize the Lazy<T> instance.

To specify the LazyThreadSafetyMode.PublicationOnly mode, use the Lazy<T>(Func<T>, LazyThreadSafetyMode) or Lazy<T>(LazyThreadSafetyMode) constructor.

Exceptions that are thrown by `valueFactory` are cached. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T>(Func<T>, LazyThreadSafetyMode)

Initializes a new instance of the Lazy<T> class that uses the specified initialization function and thread-safety mode.

C#

```
public Lazy (Func<T> valueFactory, System.Threading.LazyThreadSafetyMode
mode);
```

Parameters

`valueFactory` Func<T>

The delegate that is invoked to produce the lazily initialized value when it is needed.

`mode` LazyThreadSafetyMode

One of the enumeration values that specifies the thread safety mode.

Exceptions

ArgumentOutOfRangeException

`mode` contains an invalid value.

ArgumentNullException

`valueFactory` is `null`.

Examples

The following example demonstrates the use of this constructor to create a lazy initializer that enables multiple threads to race to create an object lazily. Multiple threads might succeed in creating instances, but all threads use the instance that was created first. In addition, the example demonstrates that exceptions are never cached when you specify LazyThreadSafetyMode.PublicationOnly, even if initialization is performed by a function instead of by the parameterless constructor of the lazily created type.

> ⓘ **Note**
>
> For an example that demonstrates how to use this constructor in single-threaded scenarios (specifying **LazyThreadSafetyMode.None** for `mode`), see the **Lazy<T>(Boolean)** constructor. For an example that demonstrates how to use this constructor to provide locking instead of race conditions in multithreaded scenarios (specifying **LazyThreadSafetyMode.ExecutionAndPublication** for `mode`), see the **Lazy<T>()** constructor.

The example defines a `LargeObject` class that will be initialized lazily by any of several threads. The four key sections of code illustrate the creation of the initializer, the actual initialization, the initialization function, and the constructor and finalizer of the `LargeObject` class. At the beginning of the `Main` method, the example creates the Lazy<T> object that performs lazy initialization of the `LargeObject`:

```C#
lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,
                                        LazyThreadSafetyMode.PublicationOnly);
```

The lazy initializer uses a function to perform the initialization. In this case, a function is required because there is no parameterless constructor for the `LargeObject` class.

The example creates and starts three threads that block on a ManualResetEvent object, so that the example can release the threads all at once. In the `ThreadProc` method that's used by all three threads, calling the Value property creates the `LargeObject` instance:

C#

```
LargeObject large = null;
try
{
    large = lazyLargeObject.Value;

    // The following line introduces an artificial delay to exaggerate
the race condition.
    Thread.Sleep(5);

    // IMPORTANT: Lazy initialization is thread-safe, but it doesn't
protect the
    //            object after creation. You must lock the object before
accessing it,
    //            unless the type is thread safe. (LargeObject is not
thread safe.)
    lock(large)
    {
        large.Data[0] = Thread.CurrentThread.ManagedThreadId;
        Console.WriteLine("LargeObject was initialized by thread {0};
last used by thread {1}.",
            large.InitializedBy, large.Data[0]);
    }
}
catch (ApplicationException ex)
{
    Console.WriteLine("ApplicationException: {0}", ex.Message);
}
```

In the third key section of code, the lazy initialization function is called to create the `LargeObject` instance. The function throws an exception the first time it's called:

C#

```
static int instanceCount = 0;
static LargeObject InitLargeObject()
{
    if (1 == Interlocked.Increment(ref instanceCount))
    {
        throw new ApplicationException(
            String.Format("Lazy initialization function failed on thread
{0}.",
            Thread.CurrentThread.ManagedThreadId));
    }
```

```
    return new LargeObject(Thread.CurrentThread.ManagedThreadId);
}
```

With any other LazyThreadSafetyMode setting, an unhandled exception in the initialization function would be cached. However, LazyThreadSafetyMode.PublicationOnly suppresses exception caching. The output from the example demonstrates that a subsequent attempt to initialize the object succeeds.

> **Note**
>
> The exception message usually appears after messages indicating that other threads have successfully initialized the object. This is because of the delay introduced by throwing and catching the exception.

Because the constructor for the Lazy<T> instance specified LazyThreadSafetyMode.PublicationOnly, all three threads are allowed to create `LargeObject` instances. The example demonstrates this by displaying console messages in the constructor and in the finalizer of the `LargeObject` class:

```C#
public LargeObject(int initializedBy)
{
    initBy = initializedBy;
    Console.WriteLine("Constructor: Instance initializing on thread {0}",
initBy);
}

~LargeObject()
{
    Console.WriteLine("Finalizer: Instance was initialized on {0}",
initBy);
}
```

The Lazy<T> object ensures that only one instance is used by all threads (except the thread where the initialization function throws an exception). The output from the example shows this.

> **Note**

For simplicity, this example uses a global instance of **Lazy<T>**, and all the methods are `static` (`Shared` in Visual Basic). These are not requirements for the use of lazy initialization.

C#

```
using System;
using System.Threading;

class Program
{
    static Lazy<LargeObject> lazyLargeObject = null;

    // Factory function for lazy initialization.
    static int instanceCount = 0;
    static LargeObject InitLargeObject()
    {
        if (1 == Interlocked.Increment(ref instanceCount))
        {
            throw new ApplicationException(
                String.Format("Lazy initialization function failed on
thread {0}.",
                Thread.CurrentThread.ManagedThreadId));
        }
        return new LargeObject(Thread.CurrentThread.ManagedThreadId);
    }

    static void Main()
    {
        // The lazy initializer is created here. LargeObject is not
created until the
        // ThreadProc method executes.
        lazyLargeObject = new Lazy<LargeObject>(InitLargeObject,

LazyThreadSafetyMode.PublicationOnly);

        // Create and start 3 threads, passing the same blocking event to
all of them.
        ManualResetEvent startingGate = new ManualResetEvent(false);
        Thread[] threads = { new Thread(ThreadProc), new
Thread(ThreadProc), new Thread(ThreadProc) };
        foreach (Thread t in threads)
        {
            t.Start(startingGate);
        }

        // Give all 3 threads time to start and wait, then release them
all at once.
        Thread.Sleep(50);
        startingGate.Set();

        // Wait for all 3 threads to finish. (The order doesn't matter.)
```

```
        foreach (Thread t in threads)
        {
            t.Join();
        }

        Console.WriteLine(
            "\r\nThreads are complete. Running GC.Collect() to reclaim
extra instances.");

        GC.Collect();

        // Allow time for garbage collection, which happens
asynchronously.
        Thread.Sleep(100);

        Console.WriteLine("\r\nNote that only one instance of LargeObject
was used.");
        Console.WriteLine("Press Enter to end the program");
        Console.ReadLine();
    }

    static void ThreadProc(object state)
    {
        // Wait for the signal.
        ManualResetEvent waitForStart = (ManualResetEvent) state;
        waitForStart.WaitOne();

        LargeObject large = null;
        try
        {
            large = lazyLargeObject.Value;

            // The following line introduces an artificial delay to
exaggerate the race condition.
            Thread.Sleep(5);

            // IMPORTANT: Lazy initialization is thread-safe, but it
doesn't protect the
            //            object after creation. You must lock the object
before accessing it,
            //            unless the type is thread safe. (LargeObject is
not thread safe.)
            lock(large)
            {
                large.Data[0] = Thread.CurrentThread.ManagedThreadId;
                Console.WriteLine("LargeObject was initialized by thread
{0}; last used by thread {1}.",
                    large.InitializedBy, large.Data[0]);
            }
        }
        catch (ApplicationException ex)
        {
            Console.WriteLine("ApplicationException: {0}", ex.Message);
        }
    }
```

```
}

class LargeObject
{
    int initBy = -1;
    public int InitializedBy { get { return initBy; } }

    public LargeObject(int initializedBy)
    {
        initBy = initializedBy;
        Console.WriteLine("Constructor: Instance initializing on thread
{0}", initBy);
    }

    ~LargeObject()
    {
        Console.WriteLine("Finalizer: Instance was initialized on {0}",
initBy);
    }

    public long[] Data = new long[100000000];
}

/* This example produces output similar to the following:

Constructor: Instance initializing on thread 5
Constructor: Instance initializing on thread 4
ApplicationException: Lazy initialization function failed on thread 3.
LargeObject was initialized by thread 5; last used by thread 5.
LargeObject was initialized by thread 5; last used by thread 4.

Threads are complete. Running GC.Collect() to reclaim extra instances.
Finalizer: Instance was initialized on 4

Note that only one instance of LargeObject was used.
Press Enter to end the program

Finalizer: Instance was initialized on 5
 */
```

Remarks

The thread safety mode of a Lazy<T> instance describes the behavior when multiple threads try to initialize the Lazy<T> instance.

Exceptions that are thrown by `valueFactory` are cached, unless `mode` is LazyThreadSafetyMode.PublicationOnly. For more information, see the Lazy<T> class or the System.Threading.LazyThreadSafetyMode enumeration.

See also

- Lazy Initialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lazy<T>.IsValueCreated Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether a value has been created for this Lazy<T> instance.

```
C#
public bool IsValueCreated { get; }
```

Property Value

Boolean

`true` if a value has been created for this Lazy<T> instance; otherwise, `false`.

Remarks

When lazy initialization occurs for a Lazy<T> instance, it may result in either a value being created or an exception being thrown. If an exception is thrown, subsequent behavior of the Lazy<T> instance depends on whether exception caching is in effect. If the Lazy<T> instance was created by using a constructor that does not specify an initialization function, then exception caching is not in effect. A subsequent attempt to initialize the Lazy<T> might succeed, and after successful initialization the IsValueCreated property returns `true`. If the Lazy<T> instance was created with an initialization function (specified by the `valueFactory` parameter of the Lazy<T> constructor), then exception caching is controlled by the thread safety mode.

- If the mode is LazyThreadSafetyMode.ExecutionAndPublication or LazyThreadSafetyMode.None, there is no second chance to initialize the Lazy<T> instance. If an exception occurs and is unhandled in the initialization function, that exception is cached and rethrown on subsequent accesses of the Lazy<T>.Value property. No value is created if an exception is thrown, so in such cases IsValueCreated returns `false`.

- If the mode is LazyThreadSafetyMode.PublicationOnly, the first thread that succeeds in running the initialization function (or the parameterless constructor) creates the value for the Lazy<T> instance. If the initialization function throws an exception on one thread, other threads can still try to initialize the Lazy<T> instance. Until the value is created, the IsValueCreated property returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- LazyThreadSafetyMode
- Lazy Initialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lazy<T>.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the lazily initialized value of the current Lazy<T> instance.

```
C#
public T Value { get; }
```

Property Value

T

The lazily initialized value of the current Lazy<T> instance.

Exceptions

MemberAccessException

The Lazy<T> instance is initialized to use the parameterless constructor of the type that is being lazily initialized, and permissions to access the constructor are missing.

MissingMemberException

The Lazy<T> instance is initialized to use the parameterless constructor of the type that is being lazily initialized, and that type does not have a public, parameterless constructor.

InvalidOperationException

The initialization function tries to access Value on this instance.

Remarks

If the Lazy<T>.IsValueCreated property is `false`, accessing the Value property forces initialization.

In addition to the exceptions that are listed, the Value property can throw any unhandled exception that is thrown by the factory method that was passed to the `valueFactory` parameter of the Lazy<T>(Func<T>), Lazy<T>(Func<T>, LazyThreadSafetyMode), or Lazy<T>(Func<T>, Boolean) constructor.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lazy Initialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lazy<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates and returns a string representation of the Value property for this instance.

```
C#
public override string? ToString ();
```

Returns

String

The result of calling the ToString() method on the Value property for this instance, if the value has been created (that is, if the IsValueCreated property returns `true`). Otherwise, a string indicating that the value has not been created.

Exceptions

NullReferenceException

The Value property is `null`.

Remarks

Calling this method does not cause initialization.

The Lazy<T>.Value property can be `null` after lazy initialization if the factory method that was specified for the `valueFactory` parameter of the Lazy<T>(Func<T>), Lazy<T>(Func<T>, LazyThreadSafetyMode), or Lazy<T>(Func<T>, Boolean) constructor returns `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Lazy<T,TMetadata> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a lazy indirect reference to an object and its associated metadata for use by the Managed Extensibility Framework.

C#

```
public class Lazy<T,TMetadata> : Lazy<T>
```

Type Parameters

T

The type of the object referenced.

TMetadata

The type of the metadata.

Inheritance Object → Lazy<T> → Lazy<T,TMetadata>

Constructors

Expand table

Lazy<T,TMetadata>(Func<T>, TMetadata)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata that uses the specified function to get the referenced object.
Lazy<T,TMetadata>(Func<T>, TMetadata, Boolean)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value that uses the specified function to get the referenced object.
Lazy<T,TMetadata>(Func<T>, TMetadata, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode that uses the specified function to get the referenced object.

Lazy<T,TMetadata>(TMetadata)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata.
Lazy<T,TMetadata>(TMetadata, Boolean)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value.
Lazy<T,TMetadata>(TMetadata, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode.

Properties

Expand table

IsValueCreated	Gets a value that indicates whether a value has been created for this Lazy<T> instance. (Inherited from Lazy<T>)
Metadata	Gets the metadata associated with the referenced object.
Value	Gets the lazily initialized value of the current Lazy<T> instance. (Inherited from Lazy<T>)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the Value property for this instance. (Inherited from Lazy<T>)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Lazy<T,TMetadata> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Lazy<T,TMetadata> class.

Overloads

Expand table

Lazy<T,TMetadata>(TMetadata)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata.
Lazy<T,TMetadata>(Func<T>, TMetadata)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata that uses the specified function to get the referenced object.
Lazy<T,TMetadata>(TMetadata, Boolean)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value.
Lazy<T,TMetadata>(TMetadata, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode.
Lazy<T,TMetadata>(Func<T>, TMetadata, Boolean)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value that uses the specified function to get the referenced object.
Lazy<T,TMetadata>(Func<T>, TMetadata, LazyThreadSafetyMode)	Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode that uses the specified function to get the referenced object.

Lazy<T,TMetadata>(TMetadata)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata.

```
C#
public Lazy (TMetadata metadata);
```

Parameters

`metadata` TMetadata

The metadata associated with the referenced object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T,TMetadata>(Func<T>, TMetadata)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata that uses the specified function to get the referenced object.

```
C#
public Lazy (Func<T> valueFactory, TMetadata metadata);
```

Parameters

`valueFactory` Func<T>

A function that returns the referenced object.

`metadata` TMetadata

The metadata associated with the referenced object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T,TMetadata>(TMetadata, Boolean)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value.

```
C#

public Lazy (TMetadata metadata, bool isThreadSafe);
```

Parameters

`metadata` TMetadata

The metadata associated with the referenced object.

`isThreadSafe` Boolean

Indicates whether the Lazy<T,TMetadata> object that is created will be thread-safe.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T,TMetadata>(TMetadata, LazyThreadSafetyMode)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode.

C#

```
public Lazy (TMetadata metadata, System.Threading.LazyThreadSafetyMode
mode);
```

Parameters

`metadata` TMetadata

The metadata associated with the referenced object.

`mode` LazyThreadSafetyMode

The thread synchronization mode.

Remarks

See LazyThreadSafetyMode for more information on thread synchronization and Lazy<T,TMetadata>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T,TMetadata>(Func<T>, TMetadata, Boolean)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread safety value that uses the specified function to get the referenced object.

C#

```
public Lazy (Func<T> valueFactory, TMetadata metadata, bool
isThreadSafe);
```

Parameters

`valueFactory` Func<T>

A function that returns the referenced object.

`metadata` TMetadata

The metadata associated with the referenced object.

`isThreadSafe` Boolean

Indicates whether the Lazy<T,TMetadata> object that is created will be thread-safe.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Lazy<T,TMetadata>(Func<T>, TMetadata, LazyThreadSafetyMode)

Initializes a new instance of the Lazy<T,TMetadata> class with the specified metadata and thread synchronization mode that uses the specified function to get the referenced object.

C#

```
public Lazy (Func<T> valueFactory, TMetadata metadata,
System.Threading.LazyThreadSafetyMode mode);
```

Parameters

`valueFactory` Func<T>

A function that returns the referenced object.

`metadata` TMetadata

The metadata associated with the referenced object.

`mode` LazyThreadSafetyMode

The thread synchronization mode.

Remarks

See LazyThreadSafetyMode for more information on thread synchronization and Lazy<T,TMetadata>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lazy<T,TMetadata>.Metadata Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the metadata associated with the referenced object.

C#

```
public TMetadata Metadata { get; }
```

Property Value

TMetadata

The metadata associated with the referenced object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

LdapStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the Lightweight Directory Access Protocol (LDAP) scheme.

C#

```
public class LdapStyleUriParser : UriParser
```

Inheritance Object → UriParser → LdapStyleUriParser

Constructors

Expand table

LdapStyleUriParser()	Creates a customizable parser based on the Lightweight Directory Access Protocol (LDAP) scheme.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)

InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

LdapStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a customizable parser based on the Lightweight Directory Access Protocol (LDAP) scheme.

C#

```
public LdapStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

LoaderOptimization Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

An enumeration used with the LoaderOptimizationAttribute class to specify loader optimizations for an executable.

C#

```
public enum LoaderOptimization
```

Inheritance Object → ValueType → Enum → LoaderOptimization

Fields

Expand table

DisallowBindings	4	Ignored by the common language runtime.
DomainMask	3	Do not use. This mask selects the domain-related values, screening out the unused DisallowBindings flag.
MultiDomain	2	Indicates that the application will probably have many domains that use the same code, and the loader must share maximal internal resources across application domains.
MultiDomainHost	3	Indicates that the application will probably host unique code in multiple domains, and the loader must share resources across application domains only for globally available (strong-named) assemblies that have been added to the global assembly cache.
NotSpecified	0	Indicates that no optimizations for sharing internal resources are specified. If the default domain or hosting interface specified an optimization, then the loader uses that; otherwise, the loader uses SingleDomain.
SingleDomain	1	Indicates that the application will probably have a single domain, and loader must not share internal resources across application domains.

Remarks

.NET Framework only: The meaning of `MultiDomainHost` changed starting with .NET Framework version 2.0. The loader now shares resources across application domains only for assemblies that have been added to the global assembly cache.

> **Note**
>
> **.NET Framework only:** If custom code access security policy is set for the **AppDomain**, by using the **SetAppDomainPolicy** property, and the **AppDomain** is created using the `MultiDomain` flag, the effect is the same as specifying the `MultiDomainHost` flag; that is, only assemblies in the GAC are shared. When this occurs, the loader does not throw an exception and the application does not experience the performance gains associated with the `MultiDomain` flag.

For more information on assembly sharing and domain-neutral assembly loading, see Application Domains and Assemblies.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- LoaderOptimizationAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

LoaderOptimizationAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Used to set the default loader optimization policy for the main method of an executable application.

C#

```
[System.AttributeUsage(System.AttributeTargets.Method)]
public sealed class LoaderOptimizationAttribute : Attribute
```

Inheritance Object → Attribute → LoaderOptimizationAttribute

Attributes AttributeUsageAttribute

Remarks

The LoaderOptimizationAttribute can only be set on the main method for an executable application and is ignored on all other methods.

The LoaderOptimizationAttribute attribute is specified with a LoaderOptimization enumeration value, which tells the loader what type of application to optimize for: `SingleDomain`, `MultiDomain`, or `MultiDomainHost`.

This attribute is only a hint to the loader and does not affect program behavior.

> **Note**
>
> If custom code access security policy is set for the **AppDomain**, by using the **SetAppDomainPolicy** property, and the **AppDomain** is created using the **MultiDomain** flag, the effect is the same as specifying the **MultiDomainHost** flag; that is, only assemblies in the GAC are shared. When this occurs, the loader does not throw an exception and the application does not experience the performance gains associated with the **MultiDomain** flag.

Constructors

Expand table

LoaderOptimizationAttribute(Byte)	Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.
LoaderOptimizationAttribute(LoaderOptimization)	Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)
Value	Gets the current LoaderOptimization value for this instance.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object.

(Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- LoaderOptimization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

LoaderOptimizationAttribute Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the LoaderOptimizationAttribute class.

Overloads

Expand table

LoaderOptimizationAttribute(Byte)	Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.
LoaderOptimizationAttribute(LoaderOptimization)	Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.

LoaderOptimizationAttribute(Byte)

Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.

C#

```
public LoaderOptimizationAttribute (byte value);
```

Parameters

`value` Byte

A value equivalent to a LoaderOptimization constant.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

LoaderOptimizationAttribute(LoaderOptimization)

Initializes a new instance of the LoaderOptimizationAttribute class to the specified value.

C#

```
public LoaderOptimizationAttribute (LoaderOptimization value);
```

Parameters

`value` LoaderOptimization

A LoaderOptimization constant.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

LoaderOptimizationAttribute.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current LoaderOptimization value for this instance.

C#

```
public LoaderOptimization Value { get; }
```

Property Value

LoaderOptimization

A LoaderOptimization constant.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

LocalDataStoreSlot Class

Reference

Definition

Namespace: System

Assembly: System.Threading.Thread.dll

Encapsulates a memory slot to store local data. This class cannot be inherited.

```
C#
public sealed class LocalDataStoreSlot
```

Inheritance Object → LocalDataStoreSlot

Remarks

The .NET Framework provides two mechanisms for using thread local storage (TLS): thread-relative static fields, and data slots.

- Thread-relative static fields are `static` fields (`Shared` fields in Visual Basic) that are marked with the ThreadStaticAttribute attribute. They provide better performance than data slots, and enable compile-time type checking.
- Data slots are slower and more awkward to use than thread-relative static fields. Also, data is stored as type Object, so you must cast it to the correct type before using it. However, you can use data slots when you have insufficient information at compile time to allocate static fields.

For more information about using TLS, see Thread Local Storage: Thread-Relative Static Fields and Data Slots.

Similarly, the .NET Framework provides two mechanisms for using context local storage: context-relative static fields and data slots. Context-relative static fields are static fields that are marked with the ContextStaticAttribute attribute. The trade-offs between using these two mechanisms are similar to the tradeoffs between using thread-relative static fields and data slots.

The LocalDataStoreSlot structure serves as a local store memory mechanism that threads and contexts can use to store thread-specific and context-specific data,

respectively. The common language runtime allocates a multi-slot data store array to each process when it is created. The thread or context calls various functions to allocate a data slot in the data store, to store and retrieve a data value in the slot, and to free a data slot for reuse after the thread or context object expires.

The data slots are unique per thread or context; their values are not shared between the thread or context objects. Data slots can be allocated by a name or by an index number.

For more information about storing local data, see Thread or Context. The LocalDataStoreSlot class is used with methods such as Thread.AllocateNamedDataSlot, Context.AllocateNamedDataSlot, Thread.GetData, and Context.GetData; it does not have any methods of its own that you need to use.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Finalize()	Ensures that resources are freed and other cleanup operations are performed when the garbage collector reclaims the LocalDataStoreSlot object.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Thread
- Context

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

LocalDataStoreSlot.Finalize Method

Reference

Definition

Namespace: System

Assembly: System.Threading.Thread.dll

Ensures that resources are freed and other cleanup operations are performed when the garbage collector reclaims the LocalDataStoreSlot object.

C#

```
~LocalDataStoreSlot ();
```

Remarks

The garbage collector calls Finalize when the current object is ready to be finalized.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

See also

- Finalize()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

MarshalByRefObject Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Enables access to objects across application domain boundaries in applications that support remoting.

C#

```
public abstract class MarshalByRefObject
```

Inheritance Object → MarshalByRefObject

Derived IEHost.Execute.IEExecuteRemote
Microsoft.Win32.RegistryKey
System.AddIn.Pipeline.ContractBase
System.AppDomain
System.AppDomainManager
More...

Examples

This section contains two code examples. The first code example shows how to create an instance of a class in another application domain. The second code example shows a simple class that can be used for remoting.

Example 1

The following code example shows the simplest way to execute code in another application domain. The example defines a class named `Worker` that inherits MarshalByRefObject, with a method that displays the name of the application domain in which it is executing. The example creates instances of `Worker` in the default application domain and in a new application domain.

Note

The assembly that contains `Worker` must be loaded into both application domains, but it could load other assemblies that would exist only in the new application domain.

C#

```
using System;
using System.Reflection;

public class CreateInstanceWorker : MarshalByRefObject
{
    public void PrintDomain()
    {
        Console.WriteLine("Object is executing in AppDomain \"{0}\"",
            AppDomain.CurrentDomain.FriendlyName);
    }
}

class CreateInstanceAndUnwrapSourceSnippet
{
    public static void Main()
    {
        // Create an ordinary instance in the current AppDomain
        CreateInstanceWorker localWorker = new CreateInstanceWorker();
        localWorker.PrintDomain();

        // Create a new application domain, create an instance
        // of Worker in the application domain, and execute code
        // there.
        AppDomain ad = AppDomain.CreateDomain("New domain");
        CreateInstanceWorker remoteWorker = (CreateInstanceWorker)
ad.CreateInstanceAndUnwrap(
            typeof(CreateInstanceWorker).Assembly.FullName,
            "Worker");
        remoteWorker.PrintDomain();
    }
}

/* This code produces output similar to the following:

Object is executing in AppDomain "source.exe"
Object is executing in AppDomain "New domain"
 */
```

Example 2

The following example demonstrates a class derived from MarshalByRefObject that is used later in remoting.

C#

```
using System;
using System.Runtime.Remoting;

public class SetObjectUriForMarshalTest  {

    class TestClass : MarshalByRefObject {
    }

    public static void Main()  {

        TestClass obj = new TestClass();

        RemotingServices.SetObjectUriForMarshal(obj, "testUri");
        RemotingServices.Marshal(obj);

        Console.WriteLine(RemotingServices.GetObjectUri(obj));
    }
}
```

Remarks

An application domain is a partition in an operating system process where one or more applications reside. Objects in the same application domain communicate directly. Objects in different application domains communicate either by transporting copies of objects across application domain boundaries, or by using a proxy to exchange messages.

MarshalByRefObject is the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy. Objects that do not inherit from MarshalByRefObject are implicitly marshal by value. When a remote application references a marshal by value object, a copy of the object is passed across application domain boundaries.

MarshalByRefObject objects are accessed directly within the boundaries of the local application domain. The first time an application in a remote application domain accesses a MarshalByRefObject, a proxy is passed to the remote application. Subsequent calls on the proxy are marshaled back to the object residing in the local application domain.

Types must inherit from MarshalByRefObject when the type is used across application domain boundaries, and the state of the object must not be copied because the members of the object are not usable outside the application domain where they were created.

When you derive an object from MarshalByRefObject for use across application domain boundaries, you should not override any of its members, nor should you call its methods directly. The runtime recognizes that classes derived from MarshalByRefObject should be marshaled across app domain boundaries.

Constructors

Expand table

MarshalByRefObject()	Initializes a new instance of the MarshalByRefObject class.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetLifetimeService()	**Obsolete.** Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeLifetimeService()	**Obsolete.** Obtains a lifetime service object to control the lifetime policy for this instance.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
MemberwiseClone(Boolean)	Creates a shallow copy of the current MarshalByRefObject object.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MarshalByRefObject Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MarshalByRefObject class.

C#

```
protected MarshalByRefObject ();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MarshalByRefObject.GetLifetimeService Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This Remoting API is not supported and throws PlatformNotSupportedException.

Retrieves the current lifetime service object that controls the lifetime policy for this instance.

C#

```
[System.Obsolete("This Remoting API is not supported and throws
PlatformNotSupportedException.", DiagnosticId="SYSLIB0010",
UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public object GetLifetimeService ();
```

Returns

Object

An object of type ILease used to control the lifetime policy for this instance.

Attributes ObsoleteAttribute

Exceptions

SecurityException

The immediate caller does not have infrastructure permission.

PlatformNotSupportedException

.NET Core and .NET 5+ only: In all cases.

Remarks

This method is marked obsolete starting in .NET 5.

For more information about lifetime services, see the LifetimeServices class.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1 *(5, 6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MarshalByRefObject.InitializeLifetimeService Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This Remoting API is not supported and throws PlatformNotSupportedException.

Obtains a lifetime service object to control the lifetime policy for this instance.

C#

```
[System.Obsolete("This Remoting API is not supported and throws
PlatformNotSupportedException.", DiagnosticId="SYSLIB0010",
UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public virtual object InitializeLifetimeService ();
```

Returns

Object

An object of type ILease used to control the lifetime policy for this instance. This is the current lifetime service object for this instance if one exists; otherwise, a new lifetime service object initialized to the value of the LeaseManagerPollTime property.

Attributes ObsoleteAttribute

Exceptions

SecurityException

The immediate caller does not have infrastructure permission.

PlatformNotSupportedException

.NET Core and .NET 5+ only: In all cases.

Examples

The following code example demonstrates creating a lease.

C#

```
public class MyClass : MarshalByRefObject
{
  public override Object InitializeLifetimeService()
  {
    ILease lease = (ILease)base.InitializeLifetimeService();
    if (lease.CurrentState == LeaseState.Initial)
    {
         lease.InitialLeaseTime = TimeSpan.FromMinutes(1);
         lease.SponsorshipTimeout = TimeSpan.FromMinutes(2);
          lease.RenewOnCallTime = TimeSpan.FromSeconds(2);
    }
      return lease;
  }
}
```

Remarks

This method is marked obsolete starting in .NET 5.

For more information about lifetime services, see the LifetimeServices class.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1 *(5, 6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

MarshalByRefObject.MemberwiseClone(Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a shallow copy of the current MarshalByRefObject object.

```
C#
protected MarshalByRefObject MemberwiseClone (bool cloneIdentity);
```

Parameters

`cloneIdentity` Boolean

`false` to delete the current MarshalByRefObject object's identity, which will cause the object to be assigned a new identity when it is marshaled across a remoting boundary. A value of `false` is usually appropriate. `true` to copy the current MarshalByRefObject object's identity to its clone, which will cause remoting client calls to be routed to the remote server object.

Returns

MarshalByRefObject

A shallow copy of the current MarshalByRefObject object.

Remarks

The MemberwiseClone(Boolean) method creates a shallow copy by creating a new MarshalByRefObject object, and then copying the nonstatic fields of the current MarshalByRefObject object to the new object. If a field is a value type, a bit-by-bit copy of the field is performed. If a field is a reference type, the reference is copied but the referred object is not; therefore, the original object and its clone refer to the same object.

For example, consider a MarshalByRefObject object called X that references objects A and B. Object B, in turn, references object C. A shallow copy of X creates new object X2 that also references objects A and B. In contrast, a deep copy of X creates a new object X2 that references the new objects A2 and B2, which are copies of A and B. B2, in turn, references the new object C2, which is a copy C. Use a class that implements the ICloneable interface to perform a deep or shallow copy of an object.

The identity of a MarshalByRefObject object is defined as the remote server object that is the target of a remoting client call. By default, the memberwise clone of a MarshalByRefObject object has the same identity as the original object, which is typically not the correct behavior for clones of server-side objects that are marshaled across a remoting boundary to the client side. Specify `false`, which is usually appropriate, to delete the identity of the clone and cause a new identity to be assigned when the clone is marshaled across a remoting boundary, or `true` to cause the clone to retain the identity of the original MarshalByRefObject object. The MemberwiseClone(Boolean) method is intended to be used by developers implementing remote server objects.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.

C#

```
public static class Math
```

Inheritance Object → Math

Examples

The following example uses several mathematical and trigonometric functions from the Math class to calculate the inner angles of a trapezoid.

C#

```
/// <summary>
/// The following class represents simple functionality of the trapezoid.
/// </summary>
using System;

namespace MathClassCS
{
    class MathTrapezoidSample
    {
        private double m_longBase;
        private double m_shortBase;
        private double m_leftLeg;
        private double m_rightLeg;

        public MathTrapezoidSample(double longbase, double shortbase, double
leftLeg, double rightLeg)
        {
            m_longBase = Math.Abs(longbase);
            m_shortBase = Math.Abs(shortbase);
            m_leftLeg = Math.Abs(leftLeg);
            m_rightLeg = Math.Abs(rightLeg);
```

```
        }

        private double GetRightSmallBase()
        {
            return (Math.Pow(m_rightLeg,2.0) - Math.Pow(m_leftLeg,2.0) +
Math.Pow(m_longBase,2.0) + Math.Pow(m_shortBase,2.0) - 2* m_shortBase *
m_longBase)/ (2*(m_longBase - m_shortBase));
        }

        public double GetHeight()
        {
            double x = GetRightSmallBase();
            return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
        }

        public double GetSquare()
        {
            return GetHeight() * m_longBase / 2.0;
        }

        public double GetLeftBaseRadianAngle()
        {
            double sinX = GetHeight()/m_leftLeg;
            return Math.Round(Math.Asin(sinX),2);
        }

        public double GetRightBaseRadianAngle()
        {
            double x = GetRightSmallBase();
            double cosX = (Math.Pow(m_rightLeg,2.0) + Math.Pow(x,2.0) -
Math.Pow(GetHeight(),2.0))/(2*x*m_rightLeg);
            return Math.Round(Math.Acos(cosX),2);
        }

        public double GetLeftBaseDegreeAngle()
        {
            double x = GetLeftBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        public double GetRightBaseDegreeAngle()
        {
            double x = GetRightBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        static void Main(string[] args)
        {
            MathTrapezoidSample trpz = new MathTrapezoidSample(20.0, 10.0,
8.0, 6.0);
            Console.WriteLine("The trapezoid's bases are 20.0 and 10.0, the
trapezoid's legs are 8.0 and 6.0");
            double h = trpz.GetHeight();
            Console.WriteLine("Trapezoid height is: " + h.ToString());
            double dxR = trpz.GetLeftBaseRadianAngle();
```

```
            Console.WriteLine("Trapezoid left base angle is: " +
dxR.ToString() + " Radians");
            double dyR = trpz.GetRightBaseRadianAngle();
            Console.WriteLine("Trapezoid right base angle is: " +
dyR.ToString() + " Radians");
            double dxD = trpz.GetLeftBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxD.ToString() + " Degrees");
            double dyD = trpz.GetRightBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dyD.ToString() + " Degrees");
        }
    }
}
```

Fields

Expand table

E	Represents the natural logarithmic base, specified by the constant, `e`.
PI	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.
Tau	Represents the number of radians in one turn, specified by the constant, τ.

Methods

Expand table

Abs(Decimal)	Returns the absolute value of a Decimal number.
Abs(Double)	Returns the absolute value of a double-precision floating-point number.
Abs(Int16)	Returns the absolute value of a 16-bit signed integer.
Abs(Int32)	Returns the absolute value of a 32-bit signed integer.
Abs(Int64)	Returns the absolute value of a 64-bit signed integer.
Abs(IntPtr)	Returns the absolute value of a native signed integer.
Abs(SByte)	Returns the absolute value of an 8-bit signed integer.
Abs(Single)	Returns the absolute value of a single-precision floating-point number.

Acos(Double)	Returns the angle whose cosine is the specified number.
Acosh(Double)	Returns the angle whose hyperbolic cosine is the specified number.
Asin(Double)	Returns the angle whose sine is the specified number.
Asinh(Double)	Returns the angle whose hyperbolic sine is the specified number.
Atan(Double)	Returns the angle whose tangent is the specified number.
Atan2(Double, Double)	Returns the angle whose tangent is the quotient of two specified numbers.
Atanh(Double)	Returns the angle whose hyperbolic tangent is the specified number.
BigMul(Int32, Int32)	Produces the full product of two 32-bit numbers.
BigMul(Int64, Int64, Int64)	Produces the full product of two 64-bit numbers.
BigMul(UInt64, UInt64, UInt64)	Produces the full product of two unsigned 64-bit numbers.
BitDecrement(Double)	Returns the largest value that compares less than a specified value.
BitIncrement(Double)	Returns the smallest value that compares greater than a specified value.
Cbrt(Double)	Returns the cube root of a specified number.
Ceiling(Decimal)	Returns the smallest integral value that is greater than or equal to the specified decimal number.
Ceiling(Double)	Returns the smallest integral value that is greater than or equal to the specified double-precision floating-point number.
Clamp(Byte, Byte, Byte)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Decimal, Decimal, Decimal)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Double, Double, Double)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int16, Int16, Int16)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int32, Int32, Int32)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int64, Int64, Int64)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(IntPtr, IntPtr, IntPtr)	Returns `value` clamped to the inclusive range of `min` and `max`.

Clamp(SByte, SByte, SByte)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Single, Single, Single)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt16, UInt16, UInt16)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt32, UInt32, UInt32)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt64, UInt64, UInt64)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UIntPtr, UIntPtr, UIntPtr)	Returns `value` clamped to the inclusive range of `min` and `max`.
CopySign(Double, Double)	Returns a value with the magnitude of `x` and the sign of `y`.
Cos(Double)	Returns the cosine of the specified angle.
Cosh(Double)	Returns the hyperbolic cosine of the specified angle.
DivRem(Byte, Byte)	Produces the quotient and the remainder of two unsigned 8-bit numbers.
DivRem(Int16, Int16)	Produces the quotient and the remainder of two signed 16-bit numbers.
DivRem(Int32, Int32)	Produces the quotient and the remainder of two signed 32-bit numbers.
DivRem(Int32, Int32, Int32)	Calculates the quotient of two 32-bit signed integers and also returns the remainder in an output parameter.
DivRem(Int64, Int64)	Produces the quotient and the remainder of two signed 64-bit numbers.
DivRem(Int64, Int64, Int64)	Calculates the quotient of two 64-bit signed integers and also returns the remainder in an output parameter.
DivRem(IntPtr, IntPtr)	Produces the quotient and the remainder of two signed native-size numbers.
DivRem(SByte, SByte)	Produces the quotient and the remainder of two signed 8-bit numbers.
DivRem(UInt16, UInt16)	Produces the quotient and the remainder of two unsigned 16-bit numbers.
DivRem(UInt32, UInt32)	Produces the quotient and the remainder of two unsigned 32-bit numbers.
DivRem(UInt64, UInt64)	Produces the quotient and the remainder of two unsigned 64-bit numbers.
DivRem(UIntPtr, UIntPtr)	Produces the quotient and the remainder of two unsigned

	native-size numbers.
Exp(Double)	Returns `e` raised to the specified power.
Floor(Decimal)	Returns the largest integral value less than or equal to the specified decimal number.
Floor(Double)	Returns the largest integral value less than or equal to the specified double-precision floating-point number.
FusedMultiplyAdd(Double, Double, Double)	Returns (x * y) + z, rounded as one ternary operation.
IEEERemainder(Double, Double)	Returns the remainder resulting from the division of a specified number by another specified number.
ILogB(Double)	Returns the base 2 integer logarithm of a specified number.
Log(Double)	Returns the natural (base `e`) logarithm of a specified number.
Log(Double, Double)	Returns the logarithm of a specified number in a specified base.
Log10(Double)	Returns the base 10 logarithm of a specified number.
Log2(Double)	Returns the base 2 logarithm of a specified number.
Max(Byte, Byte)	Returns the larger of two 8-bit unsigned integers.
Max(Decimal, Decimal)	Returns the larger of two decimal numbers.
Max(Double, Double)	Returns the larger of two double-precision floating-point numbers.
Max(Int16, Int16)	Returns the larger of two 16-bit signed integers.
Max(Int32, Int32)	Returns the larger of two 32-bit signed integers.
Max(Int64, Int64)	Returns the larger of two 64-bit signed integers.
Max(IntPtr, IntPtr)	Returns the larger of two native signed integers.
Max(SByte, SByte)	Returns the larger of two 8-bit signed integers.
Max(Single, Single)	Returns the larger of two single-precision floating-point numbers.
Max(UInt16, UInt16)	Returns the larger of two 16-bit unsigned integers.
Max(UInt32, UInt32)	Returns the larger of two 32-bit unsigned integers.
Max(UInt64, UInt64)	Returns the larger of two 64-bit unsigned integers.
Max(UIntPtr, UIntPtr)	Returns the larger of two native unsigned integers.

MaxMagnitude(Double, Double)	Returns the larger magnitude of two double-precision floating-point numbers.
Min(Byte, Byte)	Returns the smaller of two 8-bit unsigned integers.
Min(Decimal, Decimal)	Returns the smaller of two decimal numbers.
Min(Double, Double)	Returns the smaller of two double-precision floating-point numbers.
Min(Int16, Int16)	Returns the smaller of two 16-bit signed integers.
Min(Int32, Int32)	Returns the smaller of two 32-bit signed integers.
Min(Int64, Int64)	Returns the smaller of two 64-bit signed integers.
Min(IntPtr, IntPtr)	Returns the smaller of two native signed integers.
Min(SByte, SByte)	Returns the smaller of two 8-bit signed integers.
Min(Single, Single)	Returns the smaller of two single-precision floating-point numbers.
Min(UInt16, UInt16)	Returns the smaller of two 16-bit unsigned integers.
Min(UInt32, UInt32)	Returns the smaller of two 32-bit unsigned integers.
Min(UInt64, UInt64)	Returns the smaller of two 64-bit unsigned integers.
Min(UIntPtr, UIntPtr)	Returns the smaller of two native unsigned integers.
MinMagnitude(Double, Double)	Returns the smaller magnitude of two double-precision floating-point numbers.
Pow(Double, Double)	Returns a specified number raised to the specified power.
ReciprocalEstimate(Double)	Returns an estimate of the reciprocal of a specified number.
ReciprocalSqrtEstimate(Double)	Returns an estimate of the reciprocal square root of a specified number.
Round(Decimal)	Rounds a decimal value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Decimal, Int32)	Rounds a decimal value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.
Round(Decimal, Int32, MidpointRounding)	Rounds a decimal value to a specified number of fractional digits using the specified rounding convention.
Round(Decimal, MidpointRounding)	Rounds a decimal value an integer using the specified rounding convention.

Round(Double)	Rounds a double-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Double, Int32)	Rounds a double-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.
Round(Double, Int32, MidpointRounding)	Rounds a double-precision floating-point value to a specified number of fractional digits using the specified rounding convention.
Round(Double, MidpointRounding)	Rounds a double-precision floating-point value to an integer using the specified rounding convention.
ScaleB(Double, Int32)	Returns x * 2^n computed efficiently.
Sign(Decimal)	Returns an integer that indicates the sign of a decimal number.
Sign(Double)	Returns an integer that indicates the sign of a double-precision floating-point number.
Sign(Int16)	Returns an integer that indicates the sign of a 16-bit signed integer.
Sign(Int32)	Returns an integer that indicates the sign of a 32-bit signed integer.
Sign(Int64)	Returns an integer that indicates the sign of a 64-bit signed integer.
Sign(IntPtr)	Returns an integer that indicates the sign of a native sized signed integer.
Sign(SByte)	Returns an integer that indicates the sign of an 8-bit signed integer.
Sign(Single)	Returns an integer that indicates the sign of a single-precision floating-point number.
Sin(Double)	Returns the sine of the specified angle.
SinCos(Double)	Returns the sine and cosine of the specified angle.
Sinh(Double)	Returns the hyperbolic sine of the specified angle.
Sqrt(Double)	Returns the square root of a specified number.
Tan(Double)	Returns the tangent of the specified angle.
Tanh(Double)	Returns the hyperbolic tangent of the specified angle.

Truncate(Decimal)	Calculates the integral part of a specified decimal number.
Truncate(Double)	Calculates the integral part of a specified double-precision floating-point number.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.E Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the natural logarithmic base, specified by the constant, `e`.

C#

```
public const double E = 2.7182818284590451;
```

Field Value

Value = 2.7182818284590451

Double

Examples

The following example compares E with the value calculated from a power series.

C#

```
// Example for the Math.E field.
using System;

class EField
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of Math.E == {0:E16}\n" +
            "generates the following output.\n",
            Math.E );
        Console.WriteLine(
            "Define the power series PS(n) = Sum(k->0,n)[1/k!]" );
        Console.WriteLine( " (limit n->infinity)PS(n) == e" );
        Console.WriteLine(
            "Display PS(n) and Math.E - PS(n), " +
            "and stop when delta < 1.0E-15\n" );

        CalcPowerSeries();
    }
```

```
    // Approximate E with a power series.
    static void CalcPowerSeries()
    {
        double factorial = 1.0;
        double PS = 0.0;

        // Stop iterating when the series converges,
        // and prevent a runaway process.
        for( int n = 0; n < 999 && Math.Abs( Math.E - PS ) > 1.0E-15; n++ )
        {
            // Calculate a running factorial.
            if( n > 0 )
                factorial *= (double)n;

            // Calculate and display the power series.
            PS += 1.0 / factorial;
            Console.WriteLine(
                "PS({0:D2}) == {1:E16},  Math.E - PS({0:D2}) == {2:E16}",
                n, PS, Math.E - PS );
        }
    }
}

/*
This example of Math.E == 2.7182818284590451E+000
generates the following output.

Define the power series PS(n) = Sum(k->0,n)[1/k!]
 (limit n->infinity)PS(n) == e
Display PS(n) and Math.E - PS(n), and stop when delta < 1.0E-15

PS(00) == 1.0000000000000000E+000,  Math.E - PS(00) ==
1.7182818284590451E+000
PS(01) == 2.0000000000000000E+000,  Math.E - PS(01) == 7.1828182845904509E-
001
PS(02) == 2.5000000000000000E+000,  Math.E - PS(02) == 2.1828182845904509E-
001
PS(03) == 2.6666666666666665E+000,  Math.E - PS(03) == 5.1615161792378572E-
002
PS(04) == 2.7083333333333330E+000,  Math.E - PS(04) == 9.9484951257120535E-
003
PS(05) == 2.7166666666666663E+000,  Math.E - PS(05) == 1.6151617923787498E-
003
PS(06) == 2.7180555555555554E+000,  Math.E - PS(06) == 2.2627290348964380E-
004
PS(07) == 2.7182539682539684E+000,  Math.E - PS(07) == 2.7860205076724043E-
005
PS(08) == 2.7182787698412700E+000,  Math.E - PS(08) == 3.0586177750535626E-
006
PS(09) == 2.7182815255731922E+000,  Math.E - PS(09) == 3.0288585284310443E-
007
PS(10) == 2.7182818011463845E+000,  Math.E - PS(10) == 2.7312660577649694E-
008
PS(11) == 2.7182818261984929E+000,  Math.E - PS(11) == 2.2605521898810821E-
```

```
009
PS(12) == 2.7182818282861687E+000,  Math.E - PS(12) == 1.7287637987806193E-
010
PS(13) == 2.7182818284467594E+000,  Math.E - PS(13) == 1.2285727990501982E-
011
PS(14) == 2.7182818284582302E+000,  Math.E - PS(14) == 8.1490370007486490E-
013
PS(15) == 2.7182818284589949E+000,  Math.E - PS(15) == 5.0182080713057076E-
014
PS(16) == 2.7182818284590429E+000,  Math.E - PS(16) == 2.2204460492503131E-
015
PS(17) == 2.7182818284590455E+000,  Math.E - PS(17) == -4.4408920985006262E-
016
*/
```

Remarks

The value of this field is 2.7182818284590451.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.PI Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.

C#

```
public const double PI = 3.1415926535897931;
```

Field Value

Value = 3.1415926535897931

Double

Examples

The following example uses PI to assist in the computation of the inner angles of a given trapezoid.

C#

```
/// <summary>
/// The following class represents simple functionality of the trapezoid.
/// </summary>
using System;

namespace MathClassCS
{
    class MathTrapezoidSample
    {
        private double m_longBase;
        private double m_shortBase;
        private double m_leftLeg;
        private double m_rightLeg;

        public MathTrapezoidSample(double longbase, double shortbase, double
leftLeg, double rightLeg)
        {
```

```
            m_longBase = Math.Abs(longbase);
            m_shortBase = Math.Abs(shortbase);
            m_leftLeg = Math.Abs(leftLeg);
            m_rightLeg = Math.Abs(rightLeg);
        }

        private double GetRightSmallBase()
        {
            return (Math.Pow(m_rightLeg,2.0) - Math.Pow(m_leftLeg,2.0) +
Math.Pow(m_longBase,2.0) + Math.Pow(m_shortBase,2.0) - 2* m_shortBase *
m_longBase)/ (2*(m_longBase - m_shortBase));
        }

        public double GetHeight()
        {
            double x = GetRightSmallBase();
            return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
        }

        public double GetSquare()
        {
            return GetHeight() * m_longBase / 2.0;
        }

        public double GetLeftBaseRadianAngle()
        {
            double sinX = GetHeight()/m_leftLeg;
            return Math.Round(Math.Asin(sinX),2);
        }

        public double GetRightBaseRadianAngle()
        {
            double x = GetRightSmallBase();
            double cosX = (Math.Pow(m_rightLeg,2.0) + Math.Pow(x,2.0) -
Math.Pow(GetHeight(),2.0))/(2*x*m_rightLeg);
            return Math.Round(Math.Acos(cosX),2);
        }

        public double GetLeftBaseDegreeAngle()
        {
            double x = GetLeftBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        public double GetRightBaseDegreeAngle()
        {
            double x = GetRightBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        static void Main(string[] args)
        {
            MathTrapezoidSample trpz = new MathTrapezoidSample(20.0, 10.0,
8.0, 6.0);
            Console.WriteLine("The trapezoid's bases are 20.0 and 10.0, the
```

```
trapezoid's legs are 8.0 and 6.0");
            double h = trpz.GetHeight();
            Console.WriteLine("Trapezoid height is: " + h.ToString());
            double dxR = trpz.GetLeftBaseRadianAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxR.ToString() + " Radians");
            double dyR = trpz.GetRightBaseRadianAngle();
            Console.WriteLine("Trapezoid right base angle is: " +
dyR.ToString() + " Radians");
            double dxD = trpz.GetLeftBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxD.ToString() + " Degrees");
            double dyD = trpz.GetRightBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dyD.ToString() + " Degrees");
        }
    }
}
```

Remarks

The value of this field is 3.1415926535897931.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Tau Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of radians in one turn, specified by the constant, τ.

C#

```
public const double Tau = 6.2831853071795862;
```

Field Value

Value* = *6.2831853071795862

Double

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Abs Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the absolute value of a specified number.

Overloads

Expand table

Abs(Decimal)	Returns the absolute value of a Decimal number.
Abs(Double)	Returns the absolute value of a double-precision floating-point number.
Abs(Int16)	Returns the absolute value of a 16-bit signed integer.
Abs(Int32)	Returns the absolute value of a 32-bit signed integer.
Abs(Int64)	Returns the absolute value of a 64-bit signed integer.
Abs(IntPtr)	Returns the absolute value of a native signed integer.
Abs(SByte)	Returns the absolute value of an 8-bit signed integer.
Abs(Single)	Returns the absolute value of a single-precision floating-point number.

Abs(Decimal)

Returns the absolute value of a Decimal number.

C#

```
public static decimal Abs (decimal value);
```

Parameters

`value` Decimal

A number that is greater than or equal to Decimal.MinValue, but less than or equal to Decimal.MaxValue.

Returns

Decimal

A decimal number, x, such that 0 ≤ x ≤ Decimal.MaxValue.

Examples

The following example uses the Abs(Decimal) method to get the absolute value of a number of Decimal values.

C#

```
decimal[] decimals = { Decimal.MaxValue, 12.45M, 0M, -19.69M,
                       Decimal.MinValue };
foreach (decimal value in decimals)
   Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");

// The example displays the following output:
//       Abs(79228162514264337593543950335) =
79228162514264337593543950335
//       Abs(12.45) = 12.45
//       Abs(0) = 0
//       Abs(-19.69) = 19.69
//       Abs(-79228162514264337593543950335) =
79228162514264337593543950335
```

Remarks

The absolute value of a Decimal is its numeric value without its sign. For example, the absolute value of both 1.2 and -1.2 is 1.2.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(Double)

Returns the absolute value of a double-precision floating-point number.

C#

```
public static double Abs (double value);
```

Parameters

`value` Double

A number that is greater than or equal to Double.MinValue, but less than or equal to Double.MaxValue.

Returns

Double

A double-precision floating-point number, x, such that 0 ≤ x ≤ Double.MaxValue.

Examples

The following example uses the Abs(Double) method to get the absolute value of a number of Double values.

C#

```
double[] doubles = { Double.MaxValue, 16.354e-17, 15.098123, 0,
                     -19.069713, -15.058e18, Double.MinValue };
foreach (double value in doubles)
   Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");

// The example displays the following output:
//       Abs(1.79769313486232E+308) = 1.79769313486232E+308
//       Abs(1.6354E-16) = 1.6354E-16
//       Abs(15.098123) = 15.098123
//       Abs(0) = 0
```

```
//       Abs(-19.069713) = 19.069713
//       Abs(-1.5058E+19) = 1.5058E+19
//       Abs(-1.79769313486232E+308) = 1.79769313486232E+308
```

Remarks

The absolute value of a Double is its numeric value without its sign. For example, the absolute value of both 1.2e03 and -1.2e03 is 1.2e03.

If `value` is equal to NegativeInfinity or PositiveInfinity, the return value is PositiveInfinity. If `value` is equal to NaN, the return value is NaN.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(Int16)

Returns the absolute value of a 16-bit signed integer.

C#

```
public static short Abs (short value);
```

Parameters

`value` Int16

A number that is greater than Int16.MinValue, but less than or equal to Int16.MaxValue.

Returns

Int16

A 16-bit signed integer, x, such that 0 ≤ x ≤ Int16.MaxValue.

Exceptions

OverflowException

`value` equals Int16.MinValue.

Examples

The following example uses the Abs(Int16) method to get the absolute value of a number of Int16 values.

C#

```
short[] values = { Int16.MaxValue, 10328, 0, -1476, Int16.MinValue };
foreach (short value in values)
{
   try {
      Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to calculate the absolute value of {0}.",
                        value);
   }
}

// The example displays the following output:
//       Abs(32767) = 32767
//       Abs(10328) = 10328
//       Abs(0) = 0
//       Abs(-1476) = 1476
//       Unable to calculate the absolute value of -32768.
```

Remarks

The absolute value of an Int16 is its numeric value without its sign. For example, the absolute value of both 123 and -123 is 123.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(Int32)

Returns the absolute value of a 32-bit signed integer.

C#

```
public static int Abs (int value);
```

Parameters

`value` Int32

A number that is greater than Int32.MinValue, but less than or equal to Int32.MaxValue.

Returns

Int32

A 32-bit signed integer, x, such that 0 ≤ x ≤ Int32.MaxValue.

Exceptions

OverflowException

`value` equals Int32.MinValue.

Examples

The following example uses the Abs(Int32) method to get the absolute value of a number of Int32 values.

C#

```
int[] values = { Int32.MaxValue, 16921, 0, -804128, Int32.MinValue };
foreach (int value in values)
{
   try {
      Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to calculate the absolute value of {0}.",
                        value);
   }
}

// The example displays the following output:
//       Abs(2147483647) = 2147483647
//       Abs(16921) = 16921
//       Abs(0) = 0
//       Abs(-804128) = 804128
//       Unable to calculate the absolute value of -2147483648.
```

Remarks

The absolute value of an Int32 is its numeric value without its sign. For example, the absolute value of both 123 and -123 is 123.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(Int64)

Returns the absolute value of a 64-bit signed integer.

```
C#
```

```
public static long Abs (long value);
```

Parameters

value Int64

A number that is greater than Int64.MinValue, but less than or equal to Int64.MaxValue.

Returns

Int64

A 64-bit signed integer, x, such that 0 ≤ x ≤ Int64.MaxValue.

Exceptions

OverflowException

`value` equals Int64.MinValue.

Examples

The following example uses the Abs(Int64) method to get the absolute value of a number of Int64 values.

C#

```
long[] values = { Int64.MaxValue, 109013, 0, -6871982, Int64.MinValue };
foreach (long value in values)
{
   try {
      Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to calculate the absolute value of {0}.",
                        value);
   }
}

// The example displays the following output:
//       Abs(9223372036854775807) = 9223372036854775807
//       Abs(109013) = 109013
//       Abs(0) = 0
//       Abs(-6871982) = 6871982
//       Unable to calculate the absolute value of -9223372036854775808.
```

Remarks

The absolute value of an Int64 is its numeric value without its sign. For example, the absolute value of both 123 and -123 is 123.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(IntPtr)

Returns the absolute value of a native signed integer.

C#

```
public static IntPtr Abs (IntPtr value);
```

Parameters

`value` nint

A number that is greater than MinValue, but less than or equal to MaxValue.

Returns

nint

A native signed integer, x, such that 0 ≤ x ≤ MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Abs(SByte)

> Important
>
> This API is not CLS-compliant.

Returns the absolute value of an 8-bit signed integer.

```
C#

[System.CLSCompliant(false)]
public static sbyte Abs (sbyte value);
```

Parameters

`value` SByte

A number that is greater than SByte.MinValue, but less than or equal to SByte.MaxValue.

Returns

SByte

An 8-bit signed integer, x, such that 0 ≤ x ≤ SByte.MaxValue.

Attributes CLSCompliantAttribute

Exceptions

OverflowException

`value` equals SByte.MinValue.

Examples

The following example uses the Abs(SByte) method to get the absolute value of a number of SByte values.

C#

```
sbyte[] values = { SByte.MaxValue, 98, 0, -32, SByte.MinValue };
foreach (sbyte value in values)
{
   try {
      Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");
   }
   catch (OverflowException) {
      Console.WriteLine("Unable to calculate the absolute value of {0}.",
                        value);
   }
}

// The example displays the following output:
//       Abs(127) = 127
//       Abs(98) = 98
//       Abs(0) = 0
//       Abs(-32) = 32
//       Unable to calculate the absolute value of -128.
```

Remarks

The absolute value of a signed byte is its numeric value without its sign. For example, the absolute value of both 12 and -12 is 12.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Abs(Single)

Returns the absolute value of a single-precision floating-point number.

```C#
public static float Abs (float value);
```

Parameters

`value` Single

A number that is greater than or equal to Single.MinValue, but less than or equal to Single.MaxValue.

Returns

Single

A single-precision floating-point number, x, such that 0 ≤ x ≤ Single.MaxValue.

Examples

The following example uses the Abs(Single) method to get the absolute value of a number of Single values.

```C#
float[] values= { Single.MaxValue, 16.354e-12F, 15.098123F, 0F,
                  -19.069713F, -15.058e17F, Single.MinValue };
foreach (float value in values)
   Console.WriteLine($"Abs({value}) = {Math.Abs(value)}");

// The example displays the following output:
//       Abs(3.402823E+38) = 3.402823E+38
//       Abs(1.6354E-11) = 1.6354E-11
//       Abs(15.09812) = 15.09812
//       Abs(0) = 0
//       Abs(-19.06971) = 19.06971
//       Abs(-1.5058E+18) = 1.5058E+18
//       Abs(-3.402823E+38) = 3.402823E+38
```

Remarks

The absolute value of a Single is its numeric value without its sign. For example, the absolute value of both 1.2e-03 and -1.2e03 is 1.2e03.

If `value` is equal to NegativeInfinity or PositiveInfinity, the return value is PositiveInfinity. If `value` is equal to NaN, the return value is NaN.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Acos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose cosine is the specified number.

```
C#

public static double Acos (double d);
```

Parameters

d Double

A number representing a cosine, where `d` must be greater than or equal to -1, but less than or equal to 1.

Returns

Double

An angle, θ, measured in radians, such that 0 ≤ θ ≤ π.

-or-

NaN if `d` < -1 or `d` > 1 or `d` equals NaN.

Examples

The following example uses the Acos method to assist in the computation of the inner angles of a given trapezoid.

```
C#

/// <summary>
/// The following class represents simple functionality of the trapezoid.
/// </summary>
using System;
```

```
namespace MathClassCS
{
    class MathTrapezoidSample
    {
        private double m_longBase;
        private double m_shortBase;
        private double m_leftLeg;
        private double m_rightLeg;

        public MathTrapezoidSample(double longbase, double shortbase, double
leftLeg, double rightLeg)
        {
            m_longBase = Math.Abs(longbase);
            m_shortBase = Math.Abs(shortbase);
            m_leftLeg = Math.Abs(leftLeg);
            m_rightLeg = Math.Abs(rightLeg);
        }

        private double GetRightSmallBase()
        {
            return (Math.Pow(m_rightLeg,2.0) - Math.Pow(m_leftLeg,2.0) +
Math.Pow(m_longBase,2.0) + Math.Pow(m_shortBase,2.0) - 2* m_shortBase *
m_longBase)/ (2*(m_longBase - m_shortBase));
        }

        public double GetHeight()
        {
            double x = GetRightSmallBase();
            return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
        }

        public double GetSquare()
        {
            return GetHeight() * m_longBase / 2.0;
        }

        public double GetLeftBaseRadianAngle()
        {
            double sinX = GetHeight()/m_leftLeg;
            return Math.Round(Math.Asin(sinX),2);
        }

        public double GetRightBaseRadianAngle()
        {
            double x = GetRightSmallBase();
            double cosX = (Math.Pow(m_rightLeg,2.0) + Math.Pow(x,2.0) -
Math.Pow(GetHeight(),2.0))/(2*x*m_rightLeg);
            return Math.Round(Math.Acos(cosX),2);
        }

        public double GetLeftBaseDegreeAngle()
        {
            double x = GetLeftBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }
```

```
        public double GetRightBaseDegreeAngle()
        {
            double x = GetRightBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        static void Main(string[] args)
        {
            MathTrapezoidSample trpz = new MathTrapezoidSample(20.0, 10.0,
8.0, 6.0);
            Console.WriteLine("The trapezoid's bases are 20.0 and 10.0, the
trapezoid's legs are 8.0 and 6.0");
            double h = trpz.GetHeight();
            Console.WriteLine("Trapezoid height is: " + h.ToString());
            double dxR = trpz.GetLeftBaseRadianAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxR.ToString() + " Radians");
            double dyR = trpz.GetRightBaseRadianAngle();
            Console.WriteLine("Trapezoid right base angle is: " +
dyR.ToString() + " Radians");
            double dxD = trpz.GetLeftBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxD.ToString() + " Degrees");
            double dyD = trpz.GetRightBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dyD.ToString() + " Degrees");
        }
    }
}
```

Remarks

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Acosh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic cosine is the specified number.

```C#
public static double Acosh (double d);
```

Parameters

d Double

A number representing a hyperbolic cosine, where `d` must be greater than or equal to 1, but less than or equal to PositiveInfinity.

Returns

Double

An angle, θ, measured in radians, such that 0 ≤ θ ≤ ∞.

-or-

NaN if `d` < 1 or `d` equals NaN.

Remarks

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Asin(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose sine is the specified number.

```
C#

public static double Asin (double d);
```

Parameters

d Double

A number representing a sine, where `d` must be greater than or equal to -1, but less than or equal to 1.

Returns

Double

An angle, θ, measured in radians, such that -π/2 ≤ θ ≤ π/2.

-or-

NaN if `d` < -1 or `d` > 1 or `d` equals NaN.

Examples

The following example uses Asin to assist in the computation of the inner angles of a given trapezoid.

```
C#

/// <summary>
/// The following class represents simple functionality of the trapezoid.
/// </summary>
using System;
```

```
namespace MathClassCS
{
    class MathTrapezoidSample
    {
        private double m_longBase;
        private double m_shortBase;
        private double m_leftLeg;
        private double m_rightLeg;

        public MathTrapezoidSample(double longbase, double shortbase, double
leftLeg, double rightLeg)
        {
            m_longBase = Math.Abs(longbase);
            m_shortBase = Math.Abs(shortbase);
            m_leftLeg = Math.Abs(leftLeg);
            m_rightLeg = Math.Abs(rightLeg);
        }

        private double GetRightSmallBase()
        {
            return (Math.Pow(m_rightLeg,2.0) - Math.Pow(m_leftLeg,2.0) +
Math.Pow(m_longBase,2.0) + Math.Pow(m_shortBase,2.0) - 2* m_shortBase *
m_longBase)/ (2*(m_longBase - m_shortBase));
        }

        public double GetHeight()
        {
            double x = GetRightSmallBase();
            return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
        }

        public double GetSquare()
        {
            return GetHeight() * m_longBase / 2.0;
        }

        public double GetLeftBaseRadianAngle()
        {
            double sinX = GetHeight()/m_leftLeg;
            return Math.Round(Math.Asin(sinX),2);
        }

        public double GetRightBaseRadianAngle()
        {
            double x = GetRightSmallBase();
            double cosX = (Math.Pow(m_rightLeg,2.0) + Math.Pow(x,2.0) -
Math.Pow(GetHeight(),2.0))/(2*x*m_rightLeg);
            return Math.Round(Math.Acos(cosX),2);
        }

        public double GetLeftBaseDegreeAngle()
        {
            double x = GetLeftBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }
```

```
        public double GetRightBaseDegreeAngle()
        {
            double x = GetRightBaseRadianAngle() * 180/ Math.PI;
            return Math.Round(x,2);
        }

        static void Main(string[] args)
        {
            MathTrapezoidSample trpz = new MathTrapezoidSample(20.0, 10.0,
8.0, 6.0);
            Console.WriteLine("The trapezoid's bases are 20.0 and 10.0, the
trapezoid's legs are 8.0 and 6.0");
            double h = trpz.GetHeight();
            Console.WriteLine("Trapezoid height is: " + h.ToString());
            double dxR = trpz.GetLeftBaseRadianAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxR.ToString() + " Radians");
            double dyR = trpz.GetRightBaseRadianAngle();
            Console.WriteLine("Trapezoid right base angle is: " +
dyR.ToString() + " Radians");
            double dxD = trpz.GetLeftBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dxD.ToString() + " Degrees");
            double dyD = trpz.GetRightBaseDegreeAngle();
            Console.WriteLine("Trapezoid left base angle is: " +
dyD.ToString() + " Degrees");
        }
    }
}
```

Remarks

A positive return value represents a counterclockwise angle from the x-axis; a negative return value represents a clockwise angle.

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Asinh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic sine is the specified number.

```
C#
public static double Asinh (double d);
```

Parameters

d Double

A number representing a hyperbolic sine, where `d` must be greater than or equal to NegativeInfinity, but less than or equal to PositiveInfinity.

Returns

Double

An angle, θ, measured in radians.

-or-

NaN if `d` equals NaN.

Remarks

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Atan(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose tangent is the specified number.

```
C#

public static double Atan (double d);
```

Parameters

d Double

A number representing a tangent.

Returns

Double

An angle, θ, measured in radians, such that -π/2 ≤ θ ≤ π/2.

-or-

NaN if `d` equals NaN, -π/2 rounded to double precision (-1.5707963267949) if `d` equals NegativeInfinity, or π/2 rounded to double precision (1.5707963267949) if `d` equals PositiveInfinity.

Examples

The following example demonstrates how to calculate the arctangent of a value and display it to the console.

```
C#

// This example demonstrates Math.Atan()
//                           Math.Atan2()
//                           Math.Tan()
using System;
```

```
class Sample
{
    public static void Main()
    {
    double x = 1.0;
    double y = 2.0;
    double angle;
    double radians;
    double result;

// Calculate the tangent of 30 degrees.
    angle = 30;
    radians = angle * (Math.PI/180);
    result = Math.Tan(radians);
    Console.WriteLine("The tangent of 30 degrees is {0}.", result);

// Calculate the arctangent of the previous tangent.
    radians = Math.Atan(result);
    angle = radians * (180/Math.PI);
    Console.WriteLine("The previous tangent is equivalent to {0} degrees.",
angle);

// Calculate the arctangent of an angle.
    String line1 = "{0}The arctangent of the angle formed by the x-axis and
";
    String line2 = "a vector to point ({0},{1}) is {2}, ";
    String line3 = "which is equivalent to {0} degrees.";

    radians = Math.Atan2(y, x);
    angle = radians * (180/Math.PI);

    Console.WriteLine(line1, Environment.NewLine);
    Console.WriteLine(line2, x, y, radians);
    Console.WriteLine(line3, angle);
    }
}
/*
This example produces the following results:

The tangent of 30 degrees is 0.577350269189626.
The previous tangent is equivalent to 30 degrees.

The arctangent of the angle formed by the x-axis and
a vector to point (1,2) is 1.10714871779409,
which is equivalent to 63.434948822922 degrees.
*/
```

Remarks

A positive return value represents a counterclockwise angle from the x-axis; a negative return value represents a clockwise angle.

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Atan2(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose tangent is the quotient of two specified numbers.

```
C#
public static double Atan2 (double y, double x);
```

Parameters

y Double

The y coordinate of a point.

x Double

The x coordinate of a point.

Returns

Double

An angle, θ, measured in radians, such that tan(θ) = `y` / `x`, where (`x`, `y`) is a point in the Cartesian plane. Observe the following:

- For (`x`, `y`) in quadrant 1, 0 < θ < π/2.
- For (`x`, `y`) in quadrant 2, π/2 < θ ≤ π.
- For (`x`, `y`) in quadrant 3, -π ≤ θ < -π/2.
- For (`x`, `y`) in quadrant 4, -π/2 < θ < 0.

For points on the boundaries of the quadrants, the return value is the following:

- If y is 0 and x is not negative, θ = 0.
- If y is 0 and x is negative, θ = π.

- If y is positive and x is 0, θ = π/2.
- If y is negative and x is 0, θ = -π/2.
- If y is 0 and x is 0, θ = 0.

If `x` or `y` is NaN, or if `x` and `y` are either PositiveInfinity or NegativeInfinity, the method returns NaN.

Examples

The following example demonstrates how to calculate the arctangent of an angle and a vector. The resulting value is displayed in the console.

C#

```
// This example demonstrates Math.Atan()
//                           Math.Atan2()
//                           Math.Tan()
using System;

class Sample
{
    public static void Main()
    {
    double x = 1.0;
    double y = 2.0;
    double angle;
    double radians;
    double result;

// Calculate the tangent of 30 degrees.
    angle = 30;
    radians = angle * (Math.PI/180);
    result = Math.Tan(radians);
    Console.WriteLine("The tangent of 30 degrees is {0}.", result);

// Calculate the arctangent of the previous tangent.
    radians = Math.Atan(result);
    angle = radians * (180/Math.PI);
    Console.WriteLine("The previous tangent is equivalent to {0} degrees.",
angle);

// Calculate the arctangent of an angle.
    String line1 = "{0}The arctangent of the angle formed by the x-axis and
";
    String line2 = "a vector to point ({0},{1}) is {2}, ";
    String line3 = "which is equivalent to {0} degrees.";

    radians = Math.Atan2(y, x);
    angle = radians * (180/Math.PI);
```

```
    Console.WriteLine(line1, Environment.NewLine);
    Console.WriteLine(line2, x, y, radians);
    Console.WriteLine(line3, angle);
    }
}
/*
This example produces the following results:

The tangent of 30 degrees is 0.577350269189626.
The previous tangent is equivalent to 30 degrees.

The arctangent of the angle formed by the x-axis and
a vector to point (1,2) is 1.10714871779409,
which is equivalent to 63.434948822922 degrees.
*/
```

Remarks

The return value is the angle in the Cartesian plane formed by the x-axis, and a vector starting from the origin, (0,0), and terminating at the point, (x,y).

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Math.Atanh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic tangent is the specified number.

```
C#
public static double Atanh (double d);
```

Parameters

d Double

A number representing a hyperbolic tangent, where `d` must be greater than or equal to -1, but less than or equal to 1.

Returns

Double

An angle, θ, measured in radians, such that -∞ < θ < -1, or 1 < θ < ∞.

-or-

NaN if `d` < -1 or `d` > 1 or `d` equals NaN.

Remarks

Multiply the return value by 180/Math.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.BigMul Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

BigMul(Int32, Int32)	Produces the full product of two 32-bit numbers.
BigMul(Int64, Int64, Int64)	Produces the full product of two 64-bit numbers.
BigMul(UInt64, UInt64, UInt64)	Produces the full product of two unsigned 64-bit numbers.

BigMul(Int32, Int32)

Produces the full product of two 32-bit numbers.

C#

```
public static long BigMul (int a, int b);
```

Parameters

a Int32

The first number to multiply.

b Int32

The second number to multiply.

Returns

Int64

The number containing the product of the specified numbers.

Examples

The following example demonstrates the use of the BigMul method to calculate the product of two integer values.

C#

```
// This example demonstrates Math.BigMul()
using System;

class Sample
{
    public static void Main()
    {
    int int1 = Int32.MaxValue;
    int int2 = Int32.MaxValue;
    long longResult;
//
    longResult = Math.BigMul(int1, int2);
    Console.WriteLine("Calculate the product of two Int32 values:");
    Console.WriteLine("{0} * {1} = {2}", int1, int2, longResult);
    }
}
/*
This example produces the following results:
Calculate the product of two Int32 values:
2147483647 * 2147483647 = 4611686014132420609
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

BigMul(Int64, Int64, Int64)

Produces the full product of two 64-bit numbers.

C#

```
public static long BigMul (long a, long b, out long low);
```

Parameters

a Int64

The first number to multiply.

b Int64

The second number to multiply.

low Int64

When this method returns, contains the low 64-bit of the product of the specified numbers.

Returns

Int64

The high 64-bit of the product of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

BigMul(UInt64, UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Produces the full product of two unsigned 64-bit numbers.

C#

```
[System.CLSCompliant(false)]
```

```
public static ulong BigMul (ulong a, ulong b, out ulong low);
```

Parameters

a UInt64

The first number to multiply.

b UInt64

The second number to multiply.

low UInt64

When this method returns, contains the low 64-bit of the product of the specified numbers.

Returns

UInt64

The high 64-bit of the product of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.BitDecrement(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the largest value that compares less than a specified value.

C#

```
public static double BitDecrement (double x);
```

Parameters

x Double

The value to decrement.

Returns

Double

The largest value that compares less than `x`.

-or-

NegativeInfinity if `x` equals NegativeInfinity.

-or-

NaN if `x` equals NaN.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.BitIncrement(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smallest value that compares greater than a specified value.

C#

```
public static double BitIncrement (double x);
```

Parameters

x Double

The value to increment.

Returns

Double

The smallest value that compares greater than `x`.

-or-

PositiveInfinity if `x` equals PositiveInfinity.

-or-

NaN if `x` equals NaN.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Cbrt(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the cube root of a specified number.

```
C#
public static double Cbrt (double d);
```

Parameters

d Double

The number whose cube root is to be found.

Returns

Double

The cube root of `d`.

-or-

NaN if `x` equals NaN.

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

See also

- Pow(Double, Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Ceiling Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smallest integral value greater than or equal to the specified number.

Overloads

Expand table

Ceiling(Decimal)	Returns the smallest integral value that is greater than or equal to the specified decimal number.
Ceiling(Double)	Returns the smallest integral value that is greater than or equal to the specified double-precision floating-point number.

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward positive infinity.

Ceiling(Decimal)

Returns the smallest integral value that is greater than or equal to the specified decimal number.

C#

```
public static decimal Ceiling (decimal d);
```

Parameters

d Decimal

A decimal number.

Returns

Decimal

The smallest integral value that is greater than or equal to `d`. Note that this method returns a Decimal instead of an integral type.

Examples

The following example illustrates the Math.Ceiling(Decimal) method and contrasts it with the Floor(Decimal) method.

C#

```
decimal[] values = {7.03m, 7.64m, 0.12m, -0.12m, -7.1m, -7.6m};
Console.WriteLine("  Value          Ceiling          Floor\n");
foreach (decimal value in values)
   Console.WriteLine("{0,7} {1,16} {2,14}",
                     value, Math.Ceiling(value), Math.Floor(value));
// The example displays the following output to the console:
//         Value          Ceiling          Floor
//
//          7.03                8              7
//          7.64                8              7
//          0.12                1              0
//         -0.12                0             -1
//          -7.1               -7             -8
//          -7.6               -7             -8
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward positive infinity. In other words, if `d` is positive, the presence of any fractional component causes `d` to be rounded to the next highest integer. If `d` is negative, the rounding operation causes any fractional component of `d` to be discarded. The operation of this method differs from the Floor(Decimal) method, which supports rounding toward negative infinity.

See also

- Round
- Floor(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Ceiling(Double)

Returns the smallest integral value that is greater than or equal to the specified double-precision floating-point number.

C#

```
public static double Ceiling (double a);
```

Parameters

a Double

A double-precision floating-point number.

Returns

Double

The smallest integral value that is greater than or equal to `a`. If `a` is equal to NaN, NegativeInfinity, or PositiveInfinity, that value is returned. Note that this method returns a Double instead of an integral type.

Examples

The following example illustrates the Math.Ceiling(Double) method and contrasts it with the Floor(Double) method.

C#

```
double[] values = {7.03, 7.64, 0.12, -0.12, -7.1, -7.6};
Console.WriteLine("  Value          Ceiling          Floor\n");
foreach (double value in values)
   Console.WriteLine("{0,7} {1,16} {2,14}",
                     value, Math.Ceiling(value), Math.Floor(value));
// The example displays the following output to the console:
//         Value          Ceiling          Floor
//
//          7.03                8              7
//          7.64                8              7
//          0.12                1              0
//         -0.12                0             -1
//          -7.1               -7             -8
//          -7.6               -7             -8
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward positive infinity. In other words, if `a` is positive, the presence of any fractional component causes `a` to be rounded to the next highest integer. If `a` is negative, the rounding operation causes any fractional component of `a` to be discarded. The operation of this method differs from the Floor(Double) method, which supports rounding toward negative infinity.

Starting with Visual Basic 15.8, the performance of Double-to-integer conversion is optimized if you pass the value returned by the `Ceiling` method to the any of the integral conversion functions, or if the Double value returned by `Ceiling` is automatically converted to an integer with Option Strict set to Off. This optimization allows code to run faster -- up to twice as fast for code that does a large number of conversions to integer types. The following example illustrates such optimized conversions:

VB

```
Dim d1 As Double = 1043.75133
Dim i1 As Integer = CInt(Math.Ceiling(d1))        ' Result: 1044

Dim d2 As Double = 7968.4136
Dim i2 As Integer = CInt(Math.Ceiling(d2))        ' Result: 7969
```

See also

- Round

- Floor(Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Clamp Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Clamp(UIntPtr, UIntPtr, UIntPtr)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt64, UInt64, UInt64)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt32, UInt32, UInt32)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(UInt16, UInt16, UInt16)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Single, Single, Single)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(SByte, SByte, SByte)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int64, Int64, Int64)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int32, Int32, Int32)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Int16, Int16, Int16)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Double, Double, Double)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(Decimal, Decimal, Decimal)	Returns `value` clamped to the inclusive range of `min` and `max`.

Clamp(Byte, Byte, Byte)	Returns `value` clamped to the inclusive range of `min` and `max`.
Clamp(IntPtr, IntPtr, IntPtr)	Returns `value` clamped to the inclusive range of `min` and `max`.

Clamp(UIntPtr, UIntPtr, UIntPtr)

> Important
>
> This API is not CLS-compliant.

Returns `value` clamped to the inclusive range of `min` and `max`.

C#

```
[System.CLSCompliant(false)]
public static UIntPtr Clamp (UIntPtr value, UIntPtr min, UIntPtr max);
```

Parameters

`value` nuint

The value to be clamped.

`min` nuint

The lower bound of the result.

`max` nuint

The upper bound of the result.

Returns

nuint

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Clamp(UInt64, UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Returns `value` clamped to the inclusive range of `min` and `max`.

C#

```
[System.CLSCompliant(false)]
public static ulong Clamp (ulong value, ulong min, ulong max);
```

Parameters

`value` UInt64

The value to be clamped.

`min` UInt64

The lower bound of the result.

`max` UInt64

The upper bound of the result.

Returns

UInt64

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(UInt32, UInt32, UInt32)

> Important
>
> This API is not CLS-compliant.

Returns `value` clamped to the inclusive range of `min` and `max`.

C#

```
[System.CLSCompliant(false)]
public static uint Clamp (uint value, uint min, uint max);
```

Parameters

`value` UInt32

The value to be clamped.

`min` UInt32

The lower bound of the result.

`max` UInt32

The upper bound of the result.

Returns

UInt32

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(UInt16, UInt16, UInt16)

> Important
>
> This API is not CLS-compliant.

Returns `value` clamped to the inclusive range of `min` and `max`.

C#

```
[System.CLSCompliant(false)]
```

```
public static ushort Clamp (ushort value, ushort min, ushort max);
```

Parameters

`value` UInt16

The value to be clamped.

`min` UInt16

The lower bound of the result.

`max` UInt16

The upper bound of the result.

Returns

UInt16

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Single, Single, Single)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static float Clamp (float value, float min, float max);
```

Parameters

`value` Single

The value to be clamped.

`min` Single

The lower bound of the result.

`max` Single

The upper bound of the result.

Returns

Single

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

-or-

NaN if `value` equals NaN.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(SByte, SByte, SByte)

> Important
>
> This API is not CLS-compliant.

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
[System.CLSCompliant(false)]
public static sbyte Clamp (sbyte value, sbyte min, sbyte max);
```

Parameters

`value` SByte

The value to be clamped.

`min` SByte

The lower bound of the result.

`max` SByte

The upper bound of the result.

Returns

SByte

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Int64, Int64, Int64)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static long Clamp (long value, long min, long max);
```

Parameters

`value` Int64

The value to be clamped.

`min` Int64

The lower bound of the result.

`max` Int64

The upper bound of the result.

Returns

Int64

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Int32, Int32, Int32)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static int Clamp (int value, int min, int max);
```

Parameters

`value` Int32

The value to be clamped.

`min` Int32

The lower bound of the result.

`max` Int32

The upper bound of the result.

Returns

Int32

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Int16, Int16, Int16)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static short Clamp (short value, short min, short max);
```

Parameters

`value` Int16

The value to be clamped.

`min` Int16

The lower bound of the result.

`max` Int16

The upper bound of the result.

Returns

Int16

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Double, Double, Double)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static double Clamp (double value, double min, double max);
```

Parameters

`value` Double

The value to be clamped.

`min` Double

The lower bound of the result.

`max` Double

The upper bound of the result.

Returns

Double

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

-or-

NaN if `value` equals NaN.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Decimal, Decimal, Decimal)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#
public static decimal Clamp (decimal value, decimal min, decimal max);
```

Parameters

`value` Decimal

The value to be clamped.

`min` Decimal

The lower bound of the result.

`max` Decimal

The upper bound of the result.

Returns

Decimal

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(Byte, Byte, Byte)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#

public static byte Clamp (byte value, byte min, byte max);
```

Parameters

`value` Byte

The value to be clamped.

`min` Byte

The lower bound of the result.

`max` Byte

The upper bound of the result.

Returns

Byte

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Clamp(IntPtr, IntPtr, IntPtr)

Returns `value` clamped to the inclusive range of `min` and `max`.

```
C#

public static IntPtr Clamp (IntPtr value, IntPtr min, IntPtr max);
```

Parameters

`value` nint

The value to be clamped.

`min` nint

The lower bound of the result.

`max` nint

The upper bound of the result.

Returns

nint

`value` if `min` ≤ `value` ≤ `max`.

-or-

`min` if `value` < `min`.

-or-

`max` if `max` < `value`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.CopySign(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value with the magnitude of `x` and the sign of `y`.

C#

```
public static double CopySign (double x, double y);
```

Parameters

x Double

A number whose magnitude is used in the result.

y Double

A number whose sign is the used in the result.

Returns

Double

A value with the magnitude of `x` and the sign of `y`.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Cos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the cosine of the specified angle.

C#

```
public static double Cos (double d);
```

Parameters

d Double

An angle, measured in radians.

Returns

Double

The cosine of `d`. If `d` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Examples

The following example uses Cos to evaluate certain trigonometric identities for selected angles.

C#

```
// Example for the trigonometric Math.Sin( double )
// and Math.Cos( double ) methods.
using System;

class SinCos
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of trigonometric " +
```

```
            "Math.Sin( double ), Math.Cos( double ), and Math.SinCos( double
)\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Convert selected values for X to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin^2(X) + cos^2(X) == 1\n" +
                           "   sin(2 * X) == 2 * sin(X) * cos(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );

        UseSineCosine(15.0);
        UseSineCosine(30.0);
        UseSineCosine(45.0);

        Console.WriteLine(
            "\nConvert selected values for X and Y to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin(X + Y) == sin(X) * cos(Y) + cos(X) *
sin(Y)" );
        Console.WriteLine( "   cos(X + Y) == cos(X) * cos(Y) - sin(X) *
sin(Y)" );

        UseTwoAngles(15.0, 30.0);
        UseTwoAngles(30.0, 45.0);

        Console.WriteLine(
            "\nWhen you have calls to sin(X) and cos(X) they \n" +
            "can be replaced with a single call to sincos(x):" );

        UseCombinedSineCosine(15.0);
        UseCombinedSineCosine(30.0);
        UseCombinedSineCosine(45.0);
    }

    // Evaluate trigonometric identities with a given angle.
    static void UseCombinedSineCosine(double degrees)
    {
        double angle = Math.PI * degrees / 180.0;
        (double sinAngle, double cosAngle) = Math.SinCos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                           Math.SinCos({0} deg) == ({1:E16},
{2:E16})",
            degrees, sinAngle, cosAngle);
        Console.WriteLine(
            "(double sin, double cos) = Math.SinCos({0} deg)",
            degrees );
        Console.WriteLine(
            "sin^2 + cos^2 == {0:E16}",
            sinAngle * sinAngle + cosAngle * cosAngle );
    }

    // Evaluate trigonometric identities with a given angle.
```

```
    static void UseSineCosine(double degrees)
    {
        double angle    = Math.PI * degrees / 180.0;
        double sinAngle = Math.Sin(angle);
        double cosAngle = Math.Cos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                            Math.Sin({0} deg) == {1:E16}\n" +
            "                            Math.Cos({0} deg) == {2:E16}",
            degrees, Math.Sin(angle), Math.Cos(angle) );
        Console.WriteLine(
            "(Math.Sin({0} deg))^2 + (Math.Cos({0} deg))^2 == {1:E16}",
            degrees, sinAngle * sinAngle + cosAngle * cosAngle );

        // Evaluate sin(2 * X) == 2 * sin(X) * cos(X).
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            2.0 * degrees, Math.Sin(2.0 * angle) );
        Console.WriteLine(
            "    2 * Math.Sin({0} deg) * Math.Cos({0} deg) == {1:E16}",
            degrees, 2.0 * sinAngle * cosAngle );

        // Evaluate cos(2 * X) == cos^2(X) - sin^2(X).
        Console.WriteLine(
            "                            Math.Cos({0} deg) == {1:E16}",
            2.0 * degrees, Math.Cos(2.0 * angle) );
        Console.WriteLine(
            "(Math.Cos({0} deg))^2 - (Math.Sin({0} deg))^2 == {1:E16}",
            degrees, cosAngle * cosAngle - sinAngle * sinAngle );
    }

    // Evaluate trigonometric identities that are functions of two angles.
    static void UseTwoAngles(double degreesX, double degreesY)
    {
        double  angleX  = Math.PI * degreesX / 180.0;
        double  angleY  = Math.PI * degreesY / 180.0;

        // Evaluate sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y).
        Console.WriteLine(
            "\n        Math.Sin({0} deg) * Math.Cos({1} deg) +\n" +
            "        Math.Cos({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Sin(angleX) * Math.Cos(angleY) +
            Math.Cos(angleX) * Math.Sin(angleY));
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Sin(angleX + angleY));

        // Evaluate cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y).
        Console.WriteLine(
            "        Math.Cos({0} deg) * Math.Cos({1} deg) -\n" +
            "        Math.Sin({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Cos(angleX) * Math.Cos(angleY) -
            Math.Sin(angleX) * Math.Sin(angleY));
        Console.WriteLine(
```

```
            "                                Math.Cos({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Cos(angleX + angleY));
    }
}

/*
This example of trigonometric Math.Sin( double ) and Math.Cos( double )
generates the following output.

Convert selected values for X to radians
and evaluate these trigonometric identities:
   sin^2(X) + cos^2(X) == 1
   sin(2 * X) == 2 * sin(X) * cos(X)
   cos(2 * X) == cos^2(X) - sin^2(X)

                            Math.Sin(15 deg) == 2.5881904510252074E-001
                            Math.Cos(15 deg) == 9.6592582628906831E-001
(Math.Sin(15 deg))^2 + (Math.Cos(15 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(30 deg) == 4.9999999999999994E-001
    2 * Math.Sin(15 deg) * Math.Cos(15 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Cos(15 deg))^2 - (Math.Sin(15 deg))^2 == 8.6602540378443871E-001

                            Math.Sin(30 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Sin(30 deg))^2 + (Math.Cos(30 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(60 deg) == 8.6602540378443860E-001
    2 * Math.Sin(30 deg) * Math.Cos(30 deg) == 8.6602540378443860E-001
                            Math.Cos(60 deg) == 5.0000000000000011E-001
(Math.Cos(30 deg))^2 - (Math.Sin(30 deg))^2 == 5.0000000000000022E-001

                            Math.Sin(45 deg) == 7.0710678118654746E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001
(Math.Sin(45 deg))^2 + (Math.Cos(45 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(90 deg) == 1.0000000000000000E+000
    2 * Math.Sin(45 deg) * Math.Cos(45 deg) == 1.0000000000000000E+000
                            Math.Cos(90 deg) == 6.1230317691118863E-017
(Math.Cos(45 deg))^2 - (Math.Sin(45 deg))^2 == 2.2204460492503131E-016

Convert selected values for X and Y to radians
and evaluate these trigonometric identities:
   sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y)
   cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y)

        Math.Sin(15 deg) * Math.Cos(30 deg) +
        Math.Cos(15 deg) * Math.Sin(30 deg) == 7.0710678118654746E-001
                            Math.Sin(45 deg) == 7.0710678118654746E-001
        Math.Cos(15 deg) * Math.Cos(30 deg) -
        Math.Sin(15 deg) * Math.Sin(30 deg) == 7.0710678118654757E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001

        Math.Sin(30 deg) * Math.Cos(45 deg) +
        Math.Cos(30 deg) * Math.Sin(45 deg) == 9.6592582628906831E-001
                            Math.Sin(75 deg) == 9.6592582628906820E-001
        Math.Cos(30 deg) * Math.Cos(45 deg) -
```

```
        Math.Sin(30 deg) * Math.Sin(45 deg) == 2.5881904510252085E-001
                           Math.Cos(75 deg) == 2.5881904510252096E-001
*/
```

Remarks

The angle, `d`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Cosh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic cosine of the specified angle.

C#

```
public static double Cosh (double value);
```

Parameters

`value` Double

An angle, measured in radians.

Returns

Double

The hyperbolic cosine of `value`. If `value` is equal to NegativeInfinity or PositiveInfinity, PositiveInfinity is returned. If `value` is equal to NaN, NaN is returned.

Examples

The following example uses Cosh to evaluate certain hyperbolic identities for selected values.

C#

```
// Example for the hyperbolic Math.Sinh( double )
// and Math.Cosh( double ) methods.
using System;

class SinhCosh
{
    public static void Main()
    {
        Console.WriteLine(
```

```
            "This example of hyperbolic Math.Sinh( double ) " +
            "and Math.Cosh( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Evaluate these hyperbolic identities " +
            "with selected values for X:" );
        Console.WriteLine(
            "   cosh^2(X) - sinh^2(X) == 1\n" +
            "   sinh(2 * X) == 2 * sinh(X) * cosh(X)" );
        Console.WriteLine( "   cosh(2 * X) == cosh^2(X) + sinh^2(X)" );

        UseSinhCosh(0.1);
        UseSinhCosh(1.2);
        UseSinhCosh(4.9);

        Console.WriteLine(
            "\nEvaluate these hyperbolic identities " +
            "with selected values for X and Y:" );
        Console.WriteLine(
            "   sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y)" );
        Console.WriteLine(
            "   cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y)" );

        UseTwoArgs(0.1, 1.2);
        UseTwoArgs(1.2, 4.9);
    }

    // Evaluate hyperbolic identities with a given argument.
    static void UseSinhCosh(double arg)
    {
        double sinhArg = Math.Sinh(arg);
        double coshArg = Math.Cosh(arg);

        // Evaluate cosh^2(X) - sinh^2(X) == 1.
        Console.WriteLine(
            "\n                        Math.Sinh({0}) == {1:E16}\n" +
            "                        Math.Cosh({0}) == {2:E16}",
            arg, Math.Sinh(arg), Math.Cosh(arg) );
        Console.WriteLine(
            "(Math.Cosh({0}))^2 - (Math.Sinh({0}))^2 == {1:E16}",
            arg, coshArg * coshArg - sinhArg * sinhArg );

        // Evaluate sinh(2 * X) == 2 * sinh(X) * cosh(X).
        Console.WriteLine(
            "                        Math.Sinh({0}) == {1:E16}",
            2.0 * arg, Math.Sinh(2.0 * arg) );
        Console.WriteLine(
            "    2 * Math.Sinh({0}) * Math.Cosh({0}) == {1:E16}",
            arg, 2.0 * sinhArg * coshArg );

        // Evaluate cosh(2 * X) == cosh^2(X) + sinh^2(X).
        Console.WriteLine(
            "                        Math.Cosh({0}) == {1:E16}",
            2.0 * arg, Math.Cosh(2.0 * arg) );
        Console.WriteLine(
```

```
            "(Math.Cosh({0}))^2 + (Math.Sinh({0}))^2 == {1:E16}",
            arg, coshArg * coshArg + sinhArg * sinhArg );
    }

    // Evaluate hyperbolic identities that are functions of two arguments.
    static void UseTwoArgs(double argX, double argY)
    {
        // Evaluate sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y).
        Console.WriteLine(
            "\n        Math.Sinh({0}) * Math.Cosh({1}) +\n" +
            "        Math.Cosh({0}) * Math.Sinh({1}) == {2:E16}",
            argX, argY, Math.Sinh(argX) * Math.Cosh(argY) +
            Math.Cosh(argX) * Math.Sinh(argY));
        Console.WriteLine(
            "                        Math.Sinh({0}) == {1:E16}",
            argX + argY, Math.Sinh(argX + argY));

        // Evaluate cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y).
        Console.WriteLine(
            "        Math.Cosh({0}) * Math.Cosh({1}) +\n" +
            "        Math.Sinh({0}) * Math.Sinh({1}) == {2:E16}",
            argX, argY, Math.Cosh(argX) * Math.Cosh(argY) +
            Math.Sinh(argX) * Math.Sinh(argY));
        Console.WriteLine(
            "                        Math.Cosh({0}) == {1:E16}",
            argX + argY, Math.Cosh(argX + argY));
    }
}

/*
This example of hyperbolic Math.Sinh( double ) and Math.Cosh( double )
generates the following output.

Evaluate these hyperbolic identities with selected values for X:
   cosh^2(X) - sinh^2(X) == 1
   sinh(2 * X) == 2 * sinh(X) * cosh(X)
   cosh(2 * X) == cosh^2(X) + sinh^2(X)

                        Math.Sinh(0.1) == 1.0016675001984403E-001
                        Math.Cosh(0.1) == 1.0050041680558035E+000
(Math.Cosh(0.1))^2 - (Math.Sinh(0.1))^2 == 9.9999999999999989E-001
                        Math.Sinh(0.2) == 2.0133600254109399E-001
    2 * Math.Sinh(0.1) * Math.Cosh(0.1) == 2.0133600254109396E-001
                        Math.Cosh(0.2) == 1.0200667556190759E+000
(Math.Cosh(0.1))^2 + (Math.Sinh(0.1))^2 == 1.0200667556190757E+000

                        Math.Sinh(1.2) == 1.5094613554121725E+000
                        Math.Cosh(1.2) == 1.8106555673243747E+000
(Math.Cosh(1.2))^2 - (Math.Sinh(1.2))^2 == 1.0000000000000000E+000
                        Math.Sinh(2.4) == 5.4662292136760939E+000
    2 * Math.Sinh(1.2) * Math.Cosh(1.2) == 5.4662292136760939E+000
                        Math.Cosh(2.4) == 5.5569471669655064E+000
(Math.Cosh(1.2))^2 + (Math.Sinh(1.2))^2 == 5.5569471669655064E+000

                        Math.Sinh(4.9) == 6.7141166550932297E+001
```

```
                          Math.Cosh(4.9) == 6.7148613134003227E+001
(Math.Cosh(4.9))^2 - (Math.Sinh(4.9))^2 == 1.0000000000000000E+000
                          Math.Sinh(9.8) == 9.0168724361884615E+003
    2 * Math.Sinh(4.9) * Math.Cosh(4.9) == 9.0168724361884615E+003
                          Math.Cosh(9.8) == 9.0168724916400624E+003
(Math.Cosh(4.9))^2 + (Math.Sinh(4.9))^2 == 9.0168724916400606E+003

Evaluate these hyperbolic identities with selected values for X and Y:
   sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y)
   cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y)

        Math.Sinh(0.1) * Math.Cosh(1.2) +
        Math.Cosh(0.1) * Math.Sinh(1.2) == 1.6983824372926155E+000
                          Math.Sinh(1.3) == 1.6983824372926160E+000
        Math.Cosh(0.1) * Math.Cosh(1.2) +
        Math.Sinh(0.1) * Math.Sinh(1.2) == 1.9709142303266281E+000
                          Math.Cosh(1.3) == 1.9709142303266285E+000

        Math.Sinh(1.2) * Math.Cosh(4.9) +
        Math.Cosh(1.2) * Math.Sinh(4.9) == 2.2292776360739879E+002
                          Math.Sinh(6.1) == 2.2292776360739885E+002
        Math.Cosh(1.2) * Math.Cosh(4.9) +
        Math.Sinh(1.2) * Math.Sinh(4.9) == 2.2293000647511826E+002
                          Math.Cosh(6.1) == 2.2293000647511832E+002
*/
```

Remarks

The angle, `value`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.DivRem Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the quotient of two numbers and also returns the remainder in an output parameter.

Overloads

Expand table

DivRem(IntPtr, IntPtr)	Produces the quotient and the remainder of two signed native-size numbers.
DivRem(Int64, Int64, Int64)	Calculates the quotient of two 64-bit signed integers and also returns the remainder in an output parameter.
DivRem(Int32, Int32, Int32)	Calculates the quotient of two 32-bit signed integers and also returns the remainder in an output parameter.
DivRem(UIntPtr, UIntPtr)	Produces the quotient and the remainder of two unsigned native-size numbers.
DivRem(UInt64, UInt64)	Produces the quotient and the remainder of two unsigned 64-bit numbers.
DivRem(UInt32, UInt32)	Produces the quotient and the remainder of two unsigned 32-bit numbers.
DivRem(Int64, Int64)	Produces the quotient and the remainder of two signed 64-bit numbers.
DivRem(SByte, SByte)	Produces the quotient and the remainder of two signed 8-bit numbers.
DivRem(Int32, Int32)	Produces the quotient and the remainder of two signed 32-bit numbers.
DivRem(Int16, Int16)	Produces the quotient and the remainder of two signed 16-bit numbers.
DivRem(Byte, Byte)	Produces the quotient and the remainder of two unsigned 8-bit numbers.
DivRem(UInt16,	Produces the quotient and the remainder of two unsigned 16-bit

UInt16)	numbers.

DivRem(IntPtr, IntPtr)

Produces the quotient and the remainder of two signed native-size numbers.

C#

```
public static (IntPtr Quotient, IntPtr Remainder) DivRem (IntPtr left,
IntPtr right);
```

Parameters

`left` nint

The dividend.

`right` nint

The divisor.

Returns

ValueTuple<nint,nint>

The quotient and the remainder of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(Int64, Int64, Int64)

Calculates the quotient of two 64-bit signed integers and also returns the remainder in an output parameter.

C#

```
public static long DivRem (long a, long b, out long result);
```

Parameters

`a` Int64

The dividend.

`b` Int64

The divisor.

`result` Int64

When this method returns, contains the remainder.

Returns

Int64

The quotient of the specified numbers.

Exceptions

DivideByZeroException

`b` is zero.

Examples

The following example demonstrates the DivRem(Int64, Int64, Int64) method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define several positive and negative dividends.
      long[] dividends = { Int64.MaxValue, 13952, 0, -14032,
                           Int64.MinValue };
      // Define one positive and one negative divisor.
      long[] divisors = { 2000, -2000 };

      foreach (long divisor in divisors)
      {
```

```
            foreach (long dividend in dividends)
            {
               long remainder;
               long quotient = Math.DivRem(dividend, divisor, out
remainder);
               Console.WriteLine(@"{0:N0} \ {1:N0} = {2:N0}, remainder
{3:N0}",
                                 dividend, divisor, quotient, remainder);
            }
         }
      }
   }
// The example displays the following output:
//    9,223,372,036,854,775,807 \ 2,000 = 4,611,686,018,427,387,
remainder 1,807
//    13,952 \ 2,000 = 6, remainder 1,952
//    0 \ 2,000 = 0, remainder 0
//    -14,032 \ 2,000 = -7, remainder -32
//    -9,223,372,036,854,775,808 \ 2,000 = -4,611,686,018,427,387,
remainder -1,808
//    9,223,372,036,854,775,807 \ -2,000 = -4,611,686,018,427,387,
remainder 1,807
//    13,952 \ -2,000 = -6, remainder 1,952
//    0 \ -2,000 = 0, remainder 0
//    -14,032 \ -2,000 = 7, remainder -32
//    -9,223,372,036,854,775,808 \ -2,000 = 4,611,686,018,427,387,
remainder -1,808
```

Remarks

The remainder value equals the result of the remainder operator.

See also

- IEEERemainder(Double, Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

DivRem(Int32, Int32, Int32)

Calculates the quotient of two 32-bit signed integers and also returns the remainder in an output parameter.

```
C#

public static int DivRem (int a, int b, out int result);
```

Parameters

a Int32

The dividend.

b Int32

The divisor.

result Int32

When this method returns, contains the remainder.

Returns

Int32

The quotient of the specified numbers.

Exceptions

DivideByZeroException

`b` is zero.

Examples

The following example demonstrates the DivRem(Int32, Int32, Int32) method.

```
C#

using System;

public class Example
{
   public static void Main()
   {
```

```
        // Define several positive and negative dividends.
        int[] dividends = { Int32.MaxValue, 13952, 0, -14032,
                                        Int32.MinValue };
        // Define one positive and one negative divisor.
        int[] divisors = { 2000, -2000 };

        foreach (int divisor in divisors)
        {
           foreach (int dividend in dividends)
           {
              int remainder;
              int quotient = Math.DivRem(dividend, divisor, out remainder);
              Console.WriteLine(@"{0:N0} \ {1:N0} = {2:N0}, remainder
{3:N0}",
                                dividend, divisor, quotient, remainder);
           }
        }
    }
}
// The example displays the following output:
//       2,147,483,647 \ 2,000 = 1,073,741, remainder 1,647
//       13,952 \ 2,000 = 6, remainder 1,952
//       0 \ 2,000 = 0, remainder 0
//       -14,032 \ 2,000 = -7, remainder -32
//       -2,147,483,648 \ 2,000 = -1,073,741, remainder -1,648
//       2,147,483,647 \ -2,000 = -1,073,741, remainder 1,647
//       13,952 \ -2,000 = -6, remainder 1,952
//       0 \ -2,000 = 0, remainder 0
//       -14,032 \ -2,000 = 7, remainder -32
//       -2,147,483,648 \ -2,000 = 1,073,741, remainder -1,648
```

Remarks

The remainder value equals the result of the remainder operator.

See also

- IEEERemainder(Double, Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	2.0, 2.1

DivRem(UIntPtr, UIntPtr)

> Important
>
> This API is not CLS-compliant.

Produces the quotient and the remainder of two unsigned native-size numbers.

C#

```
[System.CLSCompliant(false)]
public static (UIntPtr Quotient, UIntPtr Remainder) DivRem (UIntPtr left,
UIntPtr right);
```

Parameters

`left` nuint

The dividend.

`right` nuint

The divisor.

Returns

ValueTuple<nuint,nuint>

The quotient and the remainder of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Produces the quotient and the remainder of two unsigned 64-bit numbers.

C#

```
[System.CLSCompliant(false)]
public static (ulong Quotient, ulong Remainder) DivRem (ulong left, ulong
right);
```

Parameters

`left` UInt64

The dividend.

`right` UInt64

The divisor.

Returns

ValueTuple<UInt64,UInt64>

The quotient and the remainder of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(UInt32, UInt32)

> Important
>
> This API is not CLS-compliant.

Produces the quotient and the remainder of two unsigned 32-bit numbers.

C#

```
[System.CLSCompliant(false)]
public static (uint Quotient, uint Remainder) DivRem (uint left, uint
right);
```

Parameters

`left` UInt32

The dividend.

`right` UInt32

The divisor.

Returns

ValueTuple<UInt32,UInt32>

The quotient and the remainder of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(Int64, Int64)

Produces the quotient and the remainder of two signed 64-bit numbers.

```
C#

public static (long Quotient, long Remainder) DivRem (long left, long
right);
```

Parameters

`left` Int64

The dividend.

`right` Int64

The divisor.

Returns

ValueTuple<Int64,Int64>

The quotient and the remainder of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(SByte, SByte)

> Important
>
> This API is not CLS-compliant.

Produces the quotient and the remainder of two signed 8-bit numbers.

```
C#

[System.CLSCompliant(false)]
public static (sbyte Quotient, sbyte Remainder) DivRem (sbyte left, sbyte
```

```
right);
```

Parameters

left SByte

The dividend.

right SByte

The divisor.

Returns

ValueTuple<SByte,SByte>

The quotient and the remainder of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(Int32, Int32)

Produces the quotient and the remainder of two signed 32-bit numbers.

C#

```
public static (int Quotient, int Remainder) DivRem (int left, int right);
```

Parameters

left Int32

The dividend.

right Int32

The divisor.

Returns

ValueTuple<Int32,Int32>

The quotient and the remainder of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(Int16, Int16)

Produces the quotient and the remainder of two signed 16-bit numbers.

```
C#

public static (short Quotient, short Remainder) DivRem (short left, short
right);
```

Parameters

`left` Int16

The dividend.

`right` Int16

The divisor.

Returns

ValueTuple<Int16,Int16>

The quotient and the remainder of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(Byte, Byte)

Produces the quotient and the remainder of two unsigned 8-bit numbers.

```
C#

public static (byte Quotient, byte Remainder) DivRem (byte left, byte
right);
```

Parameters

`left` Byte

The dividend.

`right` Byte

The divisor.

Returns

ValueTuple<Byte,Byte>

The quotient and the remainder of the specified numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DivRem(UInt16, UInt16)

> Important
>
> This API is not CLS-compliant.

Produces the quotient and the remainder of two unsigned 16-bit numbers.

```
C#

[System.CLSCompliant(false)]
public static (ushort Quotient, ushort Remainder) DivRem (ushort left,
ushort right);
```

Parameters

`left` UInt16

The dividend.

`right` UInt16

The divisor.

Returns

ValueTuple<UInt16,UInt16>

The quotient and the remainder of the specified numbers.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Exp(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns `e` raised to the specified power.

C#

```
public static double Exp (double d);
```

Parameters

d Double

A number specifying a power.

Returns

Double

The number `e` raised to the power `d`. If `d` equals NaN or PositiveInfinity, that value is returned. If `d` equals NegativeInfinity, 0 is returned.

Examples

The following example uses Exp to evaluate certain exponential and logarithmic identities for selected values.

C#

```
// Example for the Math.Exp( double ) method.
using System;

class ExpDemo
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of Math.Exp( double ) " +
```

```
            "generates the following output.\n" );
        Console.WriteLine(
            "Evaluate [e ^ ln(X) == ln(e ^ X) == X] " +
            "with selected values for X:" );

        UseLnExp(0.1);
        UseLnExp(1.2);
        UseLnExp(4.9);
        UseLnExp(9.9);

        Console.WriteLine(
            "\nEvaluate these identities with " +
            "selected values for X and Y:" );
        Console.WriteLine( "   (e ^ X) * (e ^ Y) == e ^ (X + Y)" );
        Console.WriteLine( "   (e ^ X) ^ Y == e ^ (X * Y)" );
        Console.WriteLine( "   X ^ Y == e ^ (Y * ln(X))" );

        UseTwoArgs(0.1, 1.2);
        UseTwoArgs(1.2, 4.9);
        UseTwoArgs(4.9, 9.9);
    }

    // Evaluate logarithmic/exponential identity with a given argument.
    static void UseLnExp(double arg)
    {
        // Evaluate e ^ ln(X) == ln(e ^ X) == X.
        Console.WriteLine(
            "\n      Math.Exp(Math.Log({0})) == {1:E16}\n" +
            "      Math.Log(Math.Exp({0})) == {2:E16}",
            arg, Math.Exp(Math.Log(arg)), Math.Log(Math.Exp(arg)) );
    }

    // Evaluate exponential identities that are functions of two arguments.
    static void UseTwoArgs(double argX, double argY)
    {
        // Evaluate (e ^ X) * (e ^ Y) == e ^ (X + Y).
        Console.WriteLine(
            "\nMath.Exp({0}) * Math.Exp({1}) == {2:E16}" +
            "\n          Math.Exp({0} + {1}) == {3:E16}",
            argX, argY, Math.Exp(argX) * Math.Exp(argY),
            Math.Exp(argX + argY) );

        // Evaluate (e ^ X) ^ Y == e ^ (X * Y).
        Console.WriteLine(
            " Math.Pow(Math.Exp({0}), {1}) == {2:E16}" +
            "\n          Math.Exp({0} * {1}) == {3:E16}",
            argX, argY, Math.Pow(Math.Exp(argX), argY),
            Math.Exp(argX * argY) );

        // Evaluate X ^ Y == e ^ (Y * ln(X)).
        Console.WriteLine(
            "           Math.Pow({0}, {1}) == {2:E16}" +
            "\nMath.Exp({1} * Math.Log({0})) == {3:E16}",
            argX, argY, Math.Pow(argX, argY),
            Math.Exp(argY * Math.Log(argX)) );
```

```
    }
}

/*
This example of Math.Exp( double ) generates the following output.

Evaluate [e ^ ln(X) == ln(e ^ X) == X] with selected values for X:

      Math.Exp(Math.Log(0.1)) == 1.0000000000000001E-001
      Math.Log(Math.Exp(0.1)) == 1.0000000000000008E-001

      Math.Exp(Math.Log(1.2)) == 1.2000000000000000E+000
      Math.Log(Math.Exp(1.2)) == 1.2000000000000000E+000

      Math.Exp(Math.Log(4.9)) == 4.9000000000000012E+000
      Math.Log(Math.Exp(4.9)) == 4.9000000000000004E+000

      Math.Exp(Math.Log(9.9)) == 9.9000000000000004E+000
      Math.Log(Math.Exp(9.9)) == 9.9000000000000004E+000

Evaluate these identities with selected values for X and Y:
   (e ^ X) * (e ^ Y) == e ^ (X + Y)
   (e ^ X) ^ Y == e ^ (X * Y)
   X ^ Y == e ^ (Y * ln(X))

Math.Exp(0.1) * Math.Exp(1.2) == 3.6692966676192444E+000
          Math.Exp(0.1 + 1.2) == 3.6692966676192444E+000
 Math.Pow(Math.Exp(0.1), 1.2) == 1.1274968515793757E+000
          Math.Exp(0.1 * 1.2) == 1.1274968515793757E+000
           Math.Pow(0.1, 1.2) == 6.3095734448019331E-002
Math.Exp(1.2 * Math.Log(0.1)) == 6.3095734448019344E-002

Math.Exp(1.2) * Math.Exp(4.9) == 4.4585777008251705E+002
          Math.Exp(1.2 + 4.9) == 4.4585777008251716E+002
 Math.Pow(Math.Exp(1.2), 4.9) == 3.5780924170885260E+002
          Math.Exp(1.2 * 4.9) == 3.5780924170885277E+002
           Math.Pow(1.2, 4.9) == 2.4433636334442981E+000
Math.Exp(4.9 * Math.Log(1.2)) == 2.4433636334442981E+000

Math.Exp(4.9) * Math.Exp(9.9) == 2.6764450551890982E+006
          Math.Exp(4.9 + 9.9) == 2.6764450551891015E+006
 Math.Pow(Math.Exp(4.9), 9.9) == 1.1684908531676833E+021
          Math.Exp(4.9 * 9.9) == 1.1684908531676829E+021
           Math.Pow(4.9, 9.9) == 6.8067718210957060E+006
Math.Exp(9.9 * Math.Log(4.9)) == 6.8067718210956985E+006
*/
```

Remarks

`e` is a mathematical constant whose value is approximately 2.71828.

Use the Pow method to calculate powers of other bases.

Exp is the inverse of Log.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- E
- Pow(Double, Double)
- Log(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Floor Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the largest integral value less than or equal to the specified number.

Overloads

Expand table

Floor(Double)	Returns the largest integral value less than or equal to the specified double-precision floating-point number.
Floor(Decimal)	Returns the largest integral value less than or equal to the specified decimal number.

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward negative infinity.

Floor(Double)

Returns the largest integral value less than or equal to the specified double-precision floating-point number.

C#

```
public static double Floor (double d);
```

Parameters

d Double

A double-precision floating-point number.

Returns

Double

The largest integral value less than or equal to `d`. If `d` is equal to NaN, NegativeInfinity, or PositiveInfinity, that value is returned.

Examples

The following example illustrates the Math.Floor(Double) method and contrasts it with the Ceiling(Double) method.

C#

```
double[] values = {7.03, 7.64, 0.12, -0.12, -7.1, -7.6};
Console.WriteLine("  Value          Ceiling          Floor\n");
foreach (double value in values)
   Console.WriteLine("{0,7} {1,16} {2,14}",
                     value, Math.Ceiling(value), Math.Floor(value));
// The example displays the following output to the console:
//         Value          Ceiling          Floor
//
//          7.03                8              7
//          7.64                8              7
//          0.12                1              0
//         -0.12                0             -1
//          -7.1               -7             -8
//          -7.6               -7             -8
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward negative infinity. In other words, if `d` is positive, any fractional component is truncated. If `d` is negative, the presence of any fractional component causes it to be rounded to the smaller integer. The operation of this method differs from the Ceiling method, which supports rounding toward positive infinity.

Starting with Visual Basic 15.8, the performance of Double-to-integer conversion is optimized if you pass the value returned by the `Floor` method to the any of the integral conversion functions, or if the Double value returned by `Floor` is automatically converted to an integer with Option Strict set to Off. This optimization allows code to run faster -- up to twice as fast for code that does a large number of conversions to integer types. The following example illustrates such optimized conversions:

VB

```
Dim d1 As Double = 1043.75133
Dim i1 As Integer = CInt(Math.Floor(d1))        ' Result: 1043

Dim d2 As Double = 7968.4136
Dim i2 As Integer = CInt(Math.Floor(d2))        ' Result: 7968
```

See also

- Round
- Ceiling(Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Floor(Decimal)

Returns the largest integral value less than or equal to the specified decimal number.

C#

```
public static decimal Floor (decimal d);
```

Parameters

d Decimal

A decimal number.

Returns

Decimal

The largest integral value less than or equal to `d`. Note that the method returns an integral value of type Decimal.

Examples

The following example illustrates the Math.Floor(Decimal) method and contrasts it with the Ceiling(Decimal) method.

C#

```
decimal[] values = {7.03m, 7.64m, 0.12m, -0.12m, -7.1m, -7.6m};
Console.WriteLine("  Value          Ceiling          Floor\n");
foreach (decimal value in values)
   Console.WriteLine("{0,7} {1,16} {2,14}",
                     value, Math.Ceiling(value), Math.Floor(value));
// The example displays the following output to the console:
//         Value          Ceiling          Floor
//
//          7.03                8              7
//          7.64                8              7
//          0.12                1              0
//         -0.12                0             -1
//          -7.1               -7             -8
//          -7.6               -7             -8
```

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward negative infinity. In other words, if `d` is positive, any fractional component is truncated. If `d` is negative, the presence of any fractional component causes it to be rounded to the smaller integer. The operation of this method differs from the Ceiling method, which supports rounding toward positive infinity.

See also

- Round
- Ceiling(Decimal)
- Floor(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.FusedMultiplyAdd(Double, Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns (x * y) + z, rounded as one ternary operation.

C#

```
public static double FusedMultiplyAdd (double x, double y, double z);
```

Parameters

x Double

The number to be multiplied with `y`.

y Double

The number to be multiplied with `x`.

z Double

The number to be added to the result of `x` multiplied by `y`.

Returns

Double

(x * y) + z, rounded as one ternary operation.

Remarks

This computes `(x * y)` as if to infinite precision, adds `z` to that result as if to infinite precision, and finally rounds to the nearest representable value.

This differs from the non-fused sequence which would compute `(x * y)` as if to infinite precision, round the result to the nearest representable value, add `z` to the rounded

result as if to infinite precision, and finally round to the nearest representable value.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.IEEERemainder(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the remainder resulting from the division of a specified number by another specified number.

C#

```
public static double IEEERemainder (double x, double y);
```

Parameters

x Double

A dividend.

y Double

A divisor.

Returns

Double

A number equal to `x` - (`y` Q), where Q is the quotient of `x` / `y` rounded to the nearest integer (if `x` / `y` falls halfway between two integers, the even integer is returned).

If `x` - (`y` Q) is zero, the value +0 is returned if `x` is positive, or -0 if `x` is negative.

If `y` = 0, NaN is returned.

Examples

The following example contrasts the remainder returned by the IEEERemainder method with the remainder returned by the remainder operator.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Console.WriteLine($"{"IEEERemainder",35} {"Remainder operator",20}");
      ShowRemainders(3, 2);
      ShowRemainders(4, 2);
      ShowRemainders(10, 3);
      ShowRemainders(11, 3);
      ShowRemainders(27, 4);
      ShowRemainders(28, 5);
      ShowRemainders(17.8, 4);
      ShowRemainders(17.8, 4.1);
      ShowRemainders(-16.3, 4.1);
      ShowRemainders(17.8, -4.1);
      ShowRemainders(-17.8, -4.1);
   }

   private static void ShowRemainders(double number1, double number2)
   {
      var formula = $"{number1} / {number2} = ";
      var ieeeRemainder = Math.IEEERemainder(number1, number2);
      var remainder = number1 % number2;
      Console.WriteLine($"{formula,-16} {ieeeRemainder,18} {remainder,20}");
   }
}
// The example displays the following output:
//
//
//                        IEEERemainder   Remainder operator
// 3 / 2 =                           -1                    1
// 4 / 2 =                            0                    0
// 10 / 3 =                           1                    1
// 11 / 3 =                          -1                    2
// 27 / 4 =                          -1                    3
// 28 / 5 =                          -2                    3
// 17.8 / 4 =                       1.8                  1.8
// 17.8 / 4.1 =                     1.4                  1.4
// -16.3 / 4.1 =    0.0999999999999979                   -4
// 17.8 / -4.1 =                    1.4                  1.4
// -17.8 / -4.1 =                  -1.4                 -1.4
```

Remarks

This operation complies with the remainder operation defined in Section 5.1 of ANSI/IEEE Std 754-1985; IEEE Standard for Binary Floating-Point Arithmetic; Institute of Electrical and Electronics Engineers, Inc; 1985.

The IEEERemainder method is not the same as the remainder operator. Although both return the remainder after division, the formulas they use are different. The formula for the IEEERemainder method is:

```
IEEERemainder = dividend - (divisor * Math.Round(dividend / divisor))
```

In contrast, the formula for the remainder operator is:

```
Remainder = (Math.Abs(dividend) - (Math.Abs(divisor) *
            (Math.Floor(Math.Abs(dividend) / Math.Abs(divisor))))) *
            Math.Sign(dividend)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.ILogB(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 2 integer logarithm of a specified number.

C#

```
public static int ILogB (double x);
```

Parameters

x Double

The number whose logarithm is to be found.

Returns

Int32

One of the values in the following table.

Expand table

`x` parameter	Return value
Default	The base 2 integer log of `x`; that is, (int)log2(`x`).
Zero	Int32.MinValue
Equal to NaN or PositiveInfinity or NegativeInfinity	Int32.MaxValue

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Log Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the logarithm of a specified number.

Overloads

Expand table

Log(Double, Double)	Returns the logarithm of a specified number in a specified base.
Log(Double)	Returns the natural (base `e`) logarithm of a specified number.

Log(Double, Double)

Returns the logarithm of a specified number in a specified base.

C#

```
public static double Log (double a, double newBase);
```

Parameters

a Double

The number whose logarithm is to be found.

newBase Double

The base of the logarithm.

Returns

Double

One of the values in the following table. (+Infinity denotes PositiveInfinity, -Infinity denotes NegativeInfinity, and NaN denotes NaN.)

Expand table

a	newBase	Return value
a > 0	(0 < newBase < 1) -or- (newBase > 1)	lognewBase(a)
a < 0	(any value)	NaN
(any value)	newBase < 0	NaN
a != 1	newBase = 0	NaN
a != 1	newBase = +Infinity	NaN
a = NaN	(any value)	NaN
(any value)	newBase = NaN	NaN
(any value)	newBase = 1	NaN
a = 0	0 < newBase < 1	+Infinity
a = 0	newBase > 1	-Infinity
a = +Infinity	0 < newBase < 1	-Infinity
a = +Infinity	newBase > 1	+Infinity
a = 1	newBase = 0	0
a = 1	newBase = +Infinity	0

Examples

The following example uses Log to evaluate certain logarithmic identities for selected values.

C#

```
// Example for the Math.Log( double ) and Math.Log( double, double )
methods.
using System;

class LogDLogDD
{
    public static void Main()
    {
```

```
        Console.WriteLine(
            "This example of Math.Log( double ) and " +
            "Math.Log( double, double )\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Evaluate these identities with " +
            "selected values for X and B (base):" );
        Console.WriteLine( "   log(B)[X] == 1 / log(X)[B]" );
        Console.WriteLine( "   log(B)[X] == ln[X] / ln[B]" );
        Console.WriteLine( "   log(B)[X] == log(B)[e] * ln[X]" );

        UseBaseAndArg(0.1, 1.2);
        UseBaseAndArg(1.2, 4.9);
        UseBaseAndArg(4.9, 9.9);
        UseBaseAndArg(9.9, 0.1);
    }

    // Evaluate logarithmic identities that are functions of two
arguments.
    static void UseBaseAndArg(double argB, double argX)
    {
        // Evaluate log(B)[X] == 1 / log(X)[B].
        Console.WriteLine(
            "\n                   Math.Log({1}, {0}) == {2:E16}" +
            "\n             1.0 / Math.Log({0}, {1}) == {3:E16}",
            argB, argX, Math.Log(argX, argB),
            1.0 / Math.Log(argB, argX) );

        // Evaluate log(B)[X] == ln[X] / ln[B].
        Console.WriteLine(
            "        Math.Log({1}) / Math.Log({0}) == {2:E16}",
            argB, argX, Math.Log(argX) / Math.Log(argB) );

        // Evaluate log(B)[X] == log(B)[e] * ln[X].
        Console.WriteLine(
            "Math.Log(Math.E, {0}) * Math.Log({1}) == {2:E16}",
            argB, argX, Math.Log(Math.E, argB) * Math.Log(argX) );
    }
}

/*
This example of Math.Log( double ) and Math.Log( double, double )
generates the following output.

Evaluate these identities with selected values for X and B (base):
   log(B)[X] == 1 / log(X)[B]
   log(B)[X] == ln[X] / ln[B]
   log(B)[X] == log(B)[e] * ln[X]

                   Math.Log(1.2, 0.1) == -7.9181246047624818E-002
             1.0 / Math.Log(0.1, 1.2) == -7.9181246047624818E-002
        Math.Log(1.2) / Math.Log(0.1) == -7.9181246047624818E-002
Math.Log(Math.E, 0.1) * Math.Log(1.2) == -7.9181246047624804E-002

                   Math.Log(4.9, 1.2) == 8.7166610085093179E+000
```

```
             1.0 / Math.Log(1.2, 4.9) == 8.7166610085093161E+000
        Math.Log(4.9) / Math.Log(1.2) == 8.7166610085093179E+000
Math.Log(Math.E, 1.2) * Math.Log(4.9) == 8.7166610085093179E+000

                   Math.Log(9.9, 4.9) == 1.4425396251981288E+000
             1.0 / Math.Log(4.9, 9.9) == 1.4425396251981288E+000
        Math.Log(9.9) / Math.Log(4.9) == 1.4425396251981288E+000
Math.Log(Math.E, 4.9) * Math.Log(9.9) == 1.4425396251981288E+000

                   Math.Log(0.1, 9.9) == -1.0043839404494075E+000
             1.0 / Math.Log(9.9, 0.1) == -1.0043839404494075E+000
        Math.Log(0.1) / Math.Log(9.9) == -1.0043839404494075E+000
Math.Log(Math.E, 9.9) * Math.Log(0.1) == -1.0043839404494077E+000
*/
```

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Log(Double)

Returns the natural (base `e`) logarithm of a specified number.

C#

```
public static double Log (double d);
```

Parameters

`d` Double

The number whose logarithm is to be found.

Returns

Double

One of the values in the following table.

Expand table

`d` parameter	Return value
Positive	The natural logarithm of `d`; that is, ln `d`, or log e `d`
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Examples

The following example illustrates the Log method.

C#

```
using System;
public class Example
{
   public static void Main()
   {
      Console.WriteLine("  Evaluate this identity with selected values
for X:");
      Console.WriteLine("                              ln(x) = 1 / log[X]
(B)");
      Console.WriteLine();

      double[] XArgs = { 1.2, 4.9, 9.9, 0.1 };

      foreach (double argX in XArgs)
      {
         // Find natural log of argX.
         Console.WriteLine("                      Math.Log({0}) =
{1:E16}",
                           argX, Math.Log(argX));
```

```
        // Evaluate 1 / log[X](e).
        Console.WriteLine("            1.0 / Math.Log(e, {0}) =
{1:E16}",
                          argX, 1.0 / Math.Log(Math.E, argX));
        Console.WriteLine();
      }
   }
}
// This example displays the following output:
//         Evaluate this identity with selected values for X:
//                                   ln(x) = 1 / log[X](B)
//
//                            Math.Log(1.2) = 1.8232155679395459E-001
//                    1.0 / Math.Log(e, 1.2) = 1.8232155679395459E-001
//
//                            Math.Log(4.9) = 1.5892352051165810E+000
//                    1.0 / Math.Log(e, 4.9) = 1.5892352051165810E+000
//
//                            Math.Log(9.9) = 2.2925347571405443E+000
//                    1.0 / Math.Log(e, 9.9) = 2.2925347571405443E+000
//
//                            Math.Log(0.1) = -2.3025850929940455E+000
//                    1.0 / Math.Log(e, 0.1) = -2.3025850929940455E+000
```

Remarks

Parameter `d` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

See also

- E
- Exp(Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Log10(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 10 logarithm of a specified number.

C#

```
public static double Log10 (double d);
```

Parameters

d Double

A number whose logarithm is to be found.

Returns

Double

One of the values in the following table.

Expand table

`d` parameter	Return value
Positive	The base 10 log of `d`; that is, log 10`d`.
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Examples

The following example uses the Log10 method to return the base 10 logarithm for selected values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double[] numbers = {-1, 0, .105, .5, .798, 1, 4, 6.9, 10, 50,
                          100, 500, 1000, Double.MaxValue};

      foreach (double number in numbers)
         Console.WriteLine("The base 10 log of {0} is {1}.",
                           number, Math.Log10(number));
   }
}
// The example dislays the following output:
//       The base 10 log of -1 is NaN.
//       The base 10 log of 0 is -Infinity.
//       The base 10 log of 0.105 is -0.978810700930062.
//       The base 10 log of 0.5 is -0.301029995663981.
//       The base 10 log of 0.798 is -0.0979971086492706.
//       The base 10 log of 1 is 0.
//       The base 10 log of 4 is 0.602059991327962.
//       The base 10 log of 6.9 is 0.838849090737255.
//       The base 10 log of 10 is 1.
//       The base 10 log of 50 is 1.69897000433602.
//       The base 10 log of 100 is 2.
//       The base 10 log of 500 is 2.69897000433602.
//       The base 10 log of 1000 is 3.
//       The base 10 log of 1.79769313486232E+308 is 308.254715559917.
```

Remarks

Parameter `d` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Log2(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 2 logarithm of a specified number.

C#

```
public static double Log2 (double x);
```

Parameters

x Double

A number whose logarithm is to be found.

Returns

Double

One of the values in the following table.

Expand table

`x` parameter	Return value
Positive	The base 2 log of `x`; that is, log 2`x`.
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Max Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the larger of two specified numbers.

Overloads

Expand table

Max(UIntPtr, UIntPtr)	Returns the larger of two native unsigned integers.
Max(UInt32, UInt32)	Returns the larger of two 32-bit unsigned integers.
Max(UInt16, UInt16)	Returns the larger of two 16-bit unsigned integers.
Max(Single, Single)	Returns the larger of two single-precision floating-point numbers.
Max(SByte, SByte)	Returns the larger of two 8-bit signed integers.
Max(IntPtr, IntPtr)	Returns the larger of two native signed integers.
Max(UInt64, UInt64)	Returns the larger of two 64-bit unsigned integers.
Max(Int32, Int32)	Returns the larger of two 32-bit signed integers.
Max(Int16, Int16)	Returns the larger of two 16-bit signed integers.
Max(Double, Double)	Returns the larger of two double-precision floating-point numbers.
Max(Decimal, Decimal)	Returns the larger of two decimal numbers.
Max(Byte, Byte)	Returns the larger of two 8-bit unsigned integers.
Max(Int64, Int64)	Returns the larger of two 64-bit signed integers.

Examples

The following example demonstrates how to use the Max method to return and display the greater of two variables:

C#

```
string str = "{0}: The greater of {1,3} and {2,3} is {3}.";

byte     xByte1    = 1,    xByte2    = 51;
short    xShort1   = -2,   xShort2   = 52;
int      xInt1     = -3,   xInt2     = 53;
long     xLong1    = -4,   xLong2    = 54;
float    xSingle1  = 5.0f, xSingle2  = 55.0f;
double   xDouble1  = 6.0,  xDouble2  = 56.0;
Decimal  xDecimal1 = 7m,   xDecimal2 = 57m;

// The following types are not CLS-compliant.
sbyte    xSbyte1   = 101, xSbyte2  = 111;
ushort   xUshort1  = 102, xUshort2 = 112;
uint     xUint1    = 103, xUint2   = 113;
ulong    xUlong1   = 104, xUlong2  = 114;

Console.WriteLine("Display the greater of two values:\n");
Console.WriteLine(str, "Byte   ", xByte1, xByte2, Math.Max(xByte1, xByte2));
Console.WriteLine(str, "Int16  ", xShort1, xShort2, Math.Max(xShort1,
xShort2));
Console.WriteLine(str, "Int32  ", xInt1, xInt2, Math.Max(xInt1, xInt2));
Console.WriteLine(str, "Int64  ", xLong1, xLong2, Math.Max(xLong1, xLong2));
Console.WriteLine(str, "Single ", xSingle1, xSingle2, Math.Max(xSingle1,
xSingle2));
Console.WriteLine(str, "Double ", xDouble1, xDouble2, Math.Max(xDouble1,
xDouble2));
Console.WriteLine(str, "Decimal", xDecimal1, xDecimal2, Math.Max(xDecimal1,
xDecimal2));

Console.WriteLine("\nThe following types are not CLS-compliant.\n");
Console.WriteLine(str, "SByte  ", xSbyte1, xSbyte2, Math.Max(xSbyte1,
xSbyte2));
Console.WriteLine(str, "UInt16 ", xUshort1, xUshort2, Math.Max(xUshort1,
xUshort2));
Console.WriteLine(str, "UInt32 ", xUint1, xUint2, Math.Max(xUint1, xUint2));
Console.WriteLine(str, "UInt64 ", xUlong1, xUlong2, Math.Max(xUlong1,
xUlong2));

/*
This example produces the following results:

Display the greater of two values:

Byte   : The greater of   1 and  51 is 51.
Int16  : The greater of  -2 and  52 is 52.
Int32  : The greater of  -3 and  53 is 53.
Int64  : The greater of  -4 and  54 is 54.
Single : The greater of   5 and  55 is 55.
Double : The greater of   6 and  56 is 56.
Decimal: The greater of   7 and  57 is 57.

(The following types are not CLS-compliant.)
```

```
SByte  : The greater of 101 and 111 is 111.
UInt16 : The greater of 102 and 112 is 112.
UInt32 : The greater of 103 and 113 is 113.
UInt64 : The greater of 104 and 114 is 114.
*/
```

Max(UIntPtr, UIntPtr)

> Important
>
> This API is not CLS-compliant.

Returns the larger of two native unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static UIntPtr Max (UIntPtr val1, UIntPtr val2);
```

Parameters

`val1` nuint

The first of two native unsigned integers to compare.

`val2` nuint

The second of two native unsigned integers to compare.

Returns

nuint

Parameter `val1` or `val2`, whichever is larger.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Max(UInt32, UInt32)

> Important
>
> This API is not CLS-compliant.

Returns the larger of two 32-bit unsigned integers.

```
C#

[System.CLSCompliant(false)]
public static uint Max (uint val1, uint val2);
```

Parameters

`val1` UInt32

The first of two 32-bit unsigned integers to compare.

`val2` UInt32

The second of two 32-bit unsigned integers to compare.

Returns

UInt32

Parameter `val1` or `val2`, whichever is larger.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(UInt16, UInt16)

> Important
>
> This API is not CLS-compliant.

Returns the larger of two 16-bit unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static ushort Max (ushort val1, ushort val2);
```

Parameters

`val1` UInt16

The first of two 16-bit unsigned integers to compare.

`val2` UInt16

The second of two 16-bit unsigned integers to compare.

Returns

UInt16

Parameter `val1` or `val2`, whichever is larger.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Single, Single)

Returns the larger of two single-precision floating-point numbers.

C#

```
public static float Max (float val1, float val2);
```

Parameters

`val1` Single

The first of two single-precision floating-point numbers to compare.

`val2` Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `val1` or `val2`, whichever is larger. If `val1`, or `val2`, or both `val1` and `val2` are equal to NaN, NaN is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(SByte, SByte)

> Important
>
> This API is not CLS-compliant.

Returns the larger of two 8-bit signed integers.

C#

```
[System.CLSCompliant(false)]
public static sbyte Max (sbyte val1, sbyte val2);
```

Parameters

`val1` SByte

The first of two 8-bit signed integers to compare.

`val2` SByte

The second of two 8-bit signed integers to compare.

Returns

SByte

Parameter `val1` or `val2`, whichever is larger.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(IntPtr, IntPtr)

Returns the larger of two native signed integers.

C#

```
public static IntPtr Max (IntPtr val1, IntPtr val2);
```

Parameters

`val1` nint

The first of two native signed integers to compare.

`val2` nint

The second of two native signed integers to compare.

Returns

nint

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Max(UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Returns the larger of two 64-bit unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static ulong Max (ulong val1, ulong val2);
```

Parameters

`val1` UInt64

The first of two 64-bit unsigned integers to compare.

`val2` UInt64

The second of two 64-bit unsigned integers to compare.

Returns

UInt64

Parameter `val1` or `val2`, whichever is larger.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Max(Int32, Int32)

Returns the larger of two 32-bit signed integers.

```
C#
public static int Max (int val1, int val2);
```

Parameters

`val1` Int32

The first of two 32-bit signed integers to compare.

`val2` Int32

The second of two 32-bit signed integers to compare.

Returns

Int32

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Int16, Int16)

Returns the larger of two 16-bit signed integers.

C#

```
public static short Max (short val1, short val2);
```

Parameters

`val1` Int16

The first of two 16-bit signed integers to compare.

`val2` Int16

The second of two 16-bit signed integers to compare.

Returns

Int16

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Double, Double)

Returns the larger of two double-precision floating-point numbers.

C#

```
public static double Max (double val1, double val2);
```

Parameters

`val1` Double

The first of two double-precision floating-point numbers to compare.

`val2` Double

The second of two double-precision floating-point numbers to compare.

Returns

Double

Parameter `val1` or `val2`, whichever is larger. If `val1`, `val2`, or both `val1` and `val2` are equal to NaN, NaN is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Decimal, Decimal)

Returns the larger of two decimal numbers.

C#

```
public static decimal Max (decimal val1, decimal val2);
```

Parameters

`val1` Decimal

The first of two decimal numbers to compare.

`val2` Decimal

The second of two decimal numbers to compare.

Returns

Decimal

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Byte, Byte)

Returns the larger of two 8-bit unsigned integers.

C#

```
public static byte Max (byte val1, byte val2);
```

Parameters

`val1` Byte

The first of two 8-bit unsigned integers to compare.

`val2` Byte

The second of two 8-bit unsigned integers to compare.

Returns

Byte

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Max(Int64, Int64)

Returns the larger of two 64-bit signed integers.

C#

```
public static long Max (long val1, long val2);
```

Parameters

`val1` Int64

The first of two 64-bit signed integers to compare.

`val2` Int64

The second of two 64-bit signed integers to compare.

Returns

Int64

Parameter `val1` or `val2`, whichever is larger.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.MaxMagnitude(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the larger magnitude of two double-precision floating-point numbers.

C#

```
public static double MaxMagnitude (double x, double y);
```

Parameters

x Double

The first of two double-precision floating-point numbers to compare.

y Double

The second of two double-precision floating-point numbers to compare.

Returns

Double

Parameter `x` or `y`, whichever has the larger magnitude. If `x`, or `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Min Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smaller of two numbers.

Overloads

Expand table

Min(UInt32, UInt32)	Returns the smaller of two 32-bit unsigned integers.
Min(UInt16, UInt16)	Returns the smaller of two 16-bit unsigned integers.
Min(Single, Single)	Returns the smaller of two single-precision floating-point numbers.
Min(SByte, SByte)	Returns the smaller of two 8-bit signed integers.
Min(IntPtr, IntPtr)	Returns the smaller of two native signed integers.
Min(Double, Double)	Returns the smaller of two double-precision floating-point numbers.
Min(Int32, Int32)	Returns the smaller of two 32-bit signed integers.
Min(Int16, Int16)	Returns the smaller of two 16-bit signed integers.
Min(Decimal, Decimal)	Returns the smaller of two decimal numbers.
Min(Byte, Byte)	Returns the smaller of two 8-bit unsigned integers.
Min(UInt64, UInt64)	Returns the smaller of two 64-bit unsigned integers.
Min(Int64, Int64)	Returns the smaller of two 64-bit signed integers.
Min(UIntPtr, UIntPtr)	Returns the smaller of two native unsigned integers.

Examples

The following example demonstrates how to use the Min method to return and display the smaller of two variables:

C#

```
string str = "{0}: The lesser of {1,3} and {2,3} is {3}.";

byte     xByte1    = 1,    xByte2    = 51;
short    xShort1   = -2,   xShort2   = 52;
int      xInt1     = -3,   xInt2     = 53;
long     xLong1    = -4,   xLong2    = 54;
float    xSingle1  = 5.0f, xSingle2  = 55.0f;
double   xDouble1  = 6.0,  xDouble2  = 56.0;
Decimal  xDecimal1 = 7m,   xDecimal2 = 57m;

// The following types are not CLS-compliant.
sbyte    xSbyte1   = 101, xSbyte2  = 111;
ushort   xUshort1  = 102, xUshort2 = 112;
uint     xUint1    = 103, xUint2   = 113;
ulong    xUlong1   = 104, xUlong2  = 114;

Console.WriteLine("Display the lesser of two values:\n");
Console.WriteLine(str, "Byte   ", xByte1, xByte2, Math.Min(xByte1, xByte2));
Console.WriteLine(str, "Int16  ", xShort1, xShort2, Math.Min(xShort1,
xShort2));
Console.WriteLine(str, "Int32  ", xInt1, xInt2, Math.Min(xInt1, xInt2));
Console.WriteLine(str, "Int64  ", xLong1, xLong2, Math.Min(xLong1, xLong2));
Console.WriteLine(str, "Single ", xSingle1, xSingle2, Math.Min(xSingle1,
xSingle2));
Console.WriteLine(str, "Double ", xDouble1, xDouble2, Math.Min(xDouble1,
xDouble2));
Console.WriteLine(str, "Decimal", xDecimal1, xDecimal2, Math.Min(xDecimal1,
xDecimal2));

Console.WriteLine("\nThe following types are not CLS-compliant:\n");
Console.WriteLine(str, "SByte  ", xSbyte1, xSbyte2, Math.Min(xSbyte1,
xSbyte2));
Console.WriteLine(str, "UInt16 ", xUshort1, xUshort2, Math.Min(xUshort1,
xUshort2));
Console.WriteLine(str, "UInt32 ", xUint1, xUint2, Math.Min(xUint1, xUint2));
Console.WriteLine(str, "UInt64 ", xUlong1, xUlong2, Math.Min(xUlong1,
xUlong2));

/*
This example produces the following results:

Display the lesser of two values:

Byte   : The lesser of   1 and  51 is 1.
Int16  : The lesser of  -2 and  52 is -2.
Int32  : The lesser of  -3 and  53 is -3.
Int64  : The lesser of  -4 and  54 is -4.
Single : The lesser of   5 and  55 is 5.
Double : The lesser of   6 and  56 is 6.
Decimal: The lesser of   7 and  57 is 7.

The following types are not CLS-compliant:
```

```
SByte  : The lesser of 101 and 111 is 101.
UInt16 : The lesser of 102 and 112 is 102.
UInt32 : The lesser of 103 and 113 is 103.
UInt64 : The lesser of 104 and 114 is 104.
*/
```

Min(UInt32, UInt32)

> Important
>
> This API is not CLS-compliant.

Returns the smaller of two 32-bit unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static uint Min (uint val1, uint val2);
```

Parameters

`val1` UInt32

The first of two 32-bit unsigned integers to compare.

`val2` UInt32

The second of two 32-bit unsigned integers to compare.

Returns

UInt32

Parameter `val1` or `val2`, whichever is smaller.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(UInt16, UInt16)

> Important
>
> This API is not CLS-compliant.

Returns the smaller of two 16-bit unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static ushort Min (ushort val1, ushort val2);
```

Parameters

`val1` UInt16

The first of two 16-bit unsigned integers to compare.

`val2` UInt16

The second of two 16-bit unsigned integers to compare.

Returns

UInt16

Parameter `val1` or `val2`, whichever is smaller.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Single, Single)

Returns the smaller of two single-precision floating-point numbers.

C#

```
public static float Min (float val1, float val2);
```

Parameters

`val1` Single

The first of two single-precision floating-point numbers to compare.

`val2` Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `val1` or `val2`, whichever is smaller. If `val1`, `val2`, or both `val1` and `val2` are equal to NaN, NaN is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(SByte, SByte)

> Important
>
> This API is not CLS-compliant.

Returns the smaller of two 8-bit signed integers.

C#

```
[System.CLSCompliant(false)]
public static sbyte Min (sbyte val1, sbyte val2);
```

Parameters

`val1` SByte

The first of two 8-bit signed integers to compare.

`val2` SByte

The second of two 8-bit signed integers to compare.

Returns

SByte

Parameter `val1` or `val2`, whichever is smaller.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(IntPtr, IntPtr)

Returns the smaller of two native signed integers.

C#

```
public static IntPtr Min (IntPtr val1, IntPtr val2);
```

Parameters

`val1` nint

The first of two native signed integers to compare.

`val2` nint

The second of two native signed integers to compare.

Returns

nint

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Min(Double, Double)

Returns the smaller of two double-precision floating-point numbers.

```
C#
public static double Min (double val1, double val2);
```

Parameters

`val1` Double

The first of two double-precision floating-point numbers to compare.

`val2` Double

The second of two double-precision floating-point numbers to compare.

Returns

Double

Parameter `val1` or `val2`, whichever is smaller. If `val1`, `val2`, or both `val1` and `val2` are equal to NaN, NaN is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Int32, Int32)

Returns the smaller of two 32-bit signed integers.

```
C#
public static int Min (int val1, int val2);
```

Parameters

`val1` Int32

The first of two 32-bit signed integers to compare.

`val2` Int32

The second of two 32-bit signed integers to compare.

Returns

Int32

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Int16, Int16)

Returns the smaller of two 16-bit signed integers.

```
C#
public static short Min (short val1, short val2);
```

Parameters

`val1` Int16

The first of two 16-bit signed integers to compare.

`val2` Int16

The second of two 16-bit signed integers to compare.

Returns

Int16

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Decimal, Decimal)

Returns the smaller of two decimal numbers.

C#

```
public static decimal Min (decimal val1, decimal val2);
```

Parameters

`val1` Decimal

The first of two decimal numbers to compare.

`val2` Decimal

The second of two decimal numbers to compare.

Returns

Decimal

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Byte, Byte)

Returns the smaller of two 8-bit unsigned integers.

C#

```
public static byte Min (byte val1, byte val2);
```

Parameters

`val1` Byte

The first of two 8-bit unsigned integers to compare.

`val2` Byte

The second of two 8-bit unsigned integers to compare.

Returns

Byte

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(UInt64, UInt64)

> Important
>
> This API is not CLS-compliant.

Returns the smaller of two 64-bit unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static ulong Min (ulong val1, ulong val2);
```

Parameters

`val1` UInt64

The first of two 64-bit unsigned integers to compare.

`val2` UInt64

The second of two 64-bit unsigned integers to compare.

Returns

UInt64

Parameter `val1` or `val2`, whichever is smaller.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(Int64, Int64)

Returns the smaller of two 64-bit signed integers.

C#

```
public static long Min (long val1, long val2);
```

Parameters

`val1` Int64

The first of two 64-bit signed integers to compare.

`val2` Int64

The second of two 64-bit signed integers to compare.

Returns

Int64

Parameter `val1` or `val2`, whichever is smaller.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Min(UIntPtr, UIntPtr)

> Important
>
> This API is not CLS-compliant.

Returns the smaller of two native unsigned integers.

C#

```
[System.CLSCompliant(false)]
public static UIntPtr Min (UIntPtr val1, UIntPtr val2);
```

Parameters

`val1` nuint

The first of two native unsigned integers to compare.

`val2` nuint

The second of two native unsigned integers to compare.

Returns

nuint

Parameter `val1` or `val2`, whichever is smaller.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.MinMagnitude(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smaller magnitude of two double-precision floating-point numbers.

C#

```
public static double MinMagnitude (double x, double y);
```

Parameters

x Double

The first of two double-precision floating-point numbers to compare.

y Double

The second of two double-precision floating-point numbers to compare.

Returns

Double

Parameter `x` or `y`, whichever has the smaller magnitude. If `x`, or `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Pow(Double, Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a specified number raised to the specified power.

C#

```
public static double Pow (double x, double y);
```

Parameters

x Double

A double-precision floating-point number to be raised to a power.

y Double

A double-precision floating-point number that specifies a power.

Returns

Double

The number `x` raised to the power `y`.

Examples

The following example uses the Pow method to calculate the value that results from raising 2 to a power ranging from 0 to 32.

C#

```
int value = 2;
for (int power = 0; power <= 32; power++)
   Console.WriteLine($"{value}^{power} = {(long)Math.Pow(value, power):N0}
(0x{(long)Math.Pow(value, power):X})");

// The example displays the following output:
//     2^0 = 1 (0x1)
```

```
//      2^1 = 2 (0x2)
//      2^2 = 4 (0x4)
//      2^3 = 8 (0x8)
//      2^4 = 16 (0x10)
//      2^5 = 32 (0x20)
//      2^6 = 64 (0x40)
//      2^7 = 128 (0x80)
//      2^8 = 256 (0x100)
//      2^9 = 512 (0x200)
//      2^10 = 1,024 (0x400)
//      2^11 = 2,048 (0x800)
//      2^12 = 4,096 (0x1000)
//      2^13 = 8,192 (0x2000)
//      2^14 = 16,384 (0x4000)
//      2^15 = 32,768 (0x8000)
//      2^16 = 65,536 (0x10000)
//      2^17 = 131,072 (0x20000)
//      2^18 = 262,144 (0x40000)
//      2^19 = 524,288 (0x80000)
//      2^20 = 1,048,576 (0x100000)
//      2^21 = 2,097,152 (0x200000)
//      2^22 = 4,194,304 (0x400000)
//      2^23 = 8,388,608 (0x800000)
//      2^24 = 16,777,216 (0x1000000)
//      2^25 = 33,554,432 (0x2000000)
//      2^26 = 67,108,864 (0x4000000)
//      2^27 = 134,217,728 (0x8000000)
//      2^28 = 268,435,456 (0x10000000)
//      2^29 = 536,870,912 (0x20000000)
//      2^30 = 1,073,741,824 (0x40000000)
//      2^31 = 2,147,483,648 (0x80000000)
//      2^32 = 4,294,967,296 (0x100000000)
```

Remarks

The following table indicates the return value when various values or ranges of values are specified for the `x` and `y` parameters. For more information, see Double.PositiveInfinity, Double.NegativeInfinity, and Double.NaN.

Expand table

x	y	Return value
`NaN`	Any value except 0	`NaN`
`NaN`	0	1 (`NaN` on .NET Framework)
Any value except `NaN`	0	1

x	y	Return value
1	Any value except `NaN`	1
1	`NaN`	1 (`NaN` on .NET Framework)
Any value except 1	`NaN`	`NaN`
`NegativeInfinity`	< 0	0
`NegativeInfinity`	Positive odd integer	`NegativeInfinity`
`NegativeInfinity`	Positive but not an odd integer	`PositiveInfinity`
< 0 but not `NegativeInfinity`	Not an integer, `NegativeInfinity`, or `PositiveInfinity`	`NaN`
-1	= `NegativeInfinity` or `PositiveInfinity`	`NaN`
-1 < x < 1	`NegativeInfinity`	`PositiveInfinity`
-1 < x < 1	`PositiveInfinity`	0
< -1 or > 1	`NegativeInfinity`	0
< -1 or > 1	`PositiveInfinity`	`PositiveInfinity`
0	< 0	`PositiveInfinity`
0	> 0	0
`PositiveInfinity`	< 0	0
`PositiveInfinity`	> 0	`PositiveInfinity`

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Sqrt(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.ReciprocalEstimate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an estimate of the reciprocal of a specified number.

```
C#
public static double ReciprocalEstimate (double d);
```

Parameters

d Double

The number whose reciprocal is to be estimated.

Returns

Double

An estimate of the reciprocal of `d`.

Remarks

On ARM64 hardware this may use the `FRECPE` instruction which performs a single Newton-Raphson iteration.

On hardware without specialized support, this may just return `1.0 / d`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.ReciprocalSqrtEstimate(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an estimate of the reciprocal square root of a specified number.

C#

```
public static double ReciprocalSqrtEstimate (double d);
```

Parameters

d Double

The number whose reciprocal square root is to be estimated.

Returns

Double

An estimate of the reciprocal square root `d`.

Remarks

On ARM64 hardware this may use the `FRSQRTE` instruction which performs a single Newton-Raphson iteration.

On hardware without specialized support, this may just return `1.0 / Sqrt(d)`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rounds a value to the nearest integer or to the specified number of fractional digits.

Overloads

Expand table

Round(Double, Int32, MidpointRounding)	Rounds a double-precision floating-point value to a specified number of fractional digits using the specified rounding convention.
Round(Decimal, Int32, MidpointRounding)	Rounds a decimal value to a specified number of fractional digits using the specified rounding convention.
Round(Double, MidpointRounding)	Rounds a double-precision floating-point value to an integer using the specified rounding convention.
Round(Double, Int32)	Rounds a double-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.
Round(Decimal, Int32)	Rounds a decimal value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.
Round(Double)	Rounds a double-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Decimal)	Rounds a decimal value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Decimal, MidpointRounding)	Rounds a decimal value an integer using the specified rounding convention.

Examples

In addition to the examples in the Remarks section, this article includes examples that illustrate the following overloads of the `Math.Round` method:

Math.Round(Decimal)
Math.Round(Double)
Math.Round(Decimal, Int32)
Math.Round(Decimal, MidpointRounding)
Math.Round(Double, Int32)
Math.Round(Double, MidpointRounding)
Math.Round(Decimal, Int32, MidpointRounding)
Math.Round(Double, Int32, MidpointRounding)

Remarks

In this section:

- Which method do I call?
- Midpoint values and rounding conventions
- Rounding and precision
- Rounding and single-precision floating point values
- Examples of individual overloads

Which method do I call?

You can use the following table to select an appropriate rounding method. In addition to the `Math.Round` methods, it also includes Math.Ceiling and Math.Floor.

Expand table

To	Call
Round a number to an integer by using the rounding-to-nearest convention.	Round(Decimal) -or- Round(Double)
Round a number to an integer by using a specified rounding convention.	Round(Decimal, MidpointRounding) -or- Round(Double, MidpointRounding)
Round a number to a specified number of fractional digits by using the rounding to nearest convention.	Round(Decimal, Int32) -or- Round(Double, Int32)

To	Call
Round a number to a specified number of fractional digits by using a specified rounding convention.	Round(Decimal, Int32, MidpointRounding) -or- Round(Double, Int32, MidpointRounding)
Round a Single value to a specified number of fractional digits by using a specified rounding convention and minimizing the loss of precision.	Convert the Single to a Decimal and call Round(Decimal, Int32, MidpointRounding).
Round a number to a specified number of fractional digits while minimizing problems of precision in rounding midpoint values.	Call a rounding method that implements a "greater than or approximately equal to" comparison. See Rounding and precision.
Round a fractional value to an integer that is greater than the fractional value. For example, round 3.1 to 4.	Ceiling
Round a fractional value to an integer that is less than the fractional value. For example, round 3.9 to 3.	Floor

Midpoint values and rounding conventions

Rounding involves converting a numeric value with a specified precision to a value with less precision. For example, you can use the Round(Double) method to round a value of 3.4 to 3.0, and the Round(Double, Int32) method to round a value of 3.579 to 3.58.

In a midpoint value, the value after the least significant digit in the result is precisely half way between two numbers. For example, 3.47500 is a midpoint value if it is to be rounded to two decimal places, and 7.500 is a midpoint value if it is to be rounded to an integer. In these cases, if the round-to-nearest strategy is used, the nearest value can't be easily identified without a rounding convention.

The Round method supports two rounding conventions for handling midpoint values:

- **Rounding away from zero**

  Midpoint values are rounded to the next number away from zero. For example, 3.75 rounds to 3.8, 3.85 rounds to 3.9, -3.75 rounds to -3.8, and -3.85 rounds to -3.9. This form of rounding is represented by the MidpointRounding.AwayFromZero enumeration member.

- **Rounding to nearest even, or banker's rounding**

Midpoint values are rounded to the nearest even number. For example, both 3.75 and 3.85 round to 3.8, and both -3.75 and -3.85 round to -3.8. This form of rounding is represented by the MidpointRounding.ToEven enumeration member.

> **Note**
>
> In .NET Core 3.0 and later versions, three additional rounding strategies are available through the **MidpointRounding** enumeration. These strategies are used in all cases, not just for midpoint values as **MidpointRounding.ToEven** and **MidpointRounding.AwayFromZero** are.

Rounding away from zero is the most widely known form of rounding, while rounding to nearest even is the standard in financial and statistical operations. It conforms to IEEE Standard 754, section 4. When used in multiple rounding operations, rounding to nearest even reduces the rounding error that is caused by consistently rounding midpoint values in a single direction. In some cases, this rounding error can be significant.

The following example illustrates the bias that can result from consistently rounding midpoint values in a single direction. The example computes the true mean of an array of Decimal values, and then computes the mean when the values in the array are rounded by using the two conventions. In this example, the true mean and the mean that results when rounding to nearest are the same. However, the mean that results when rounding away from zero differs by .05 (or by 3.6%) from the true mean.

C#

```
decimal[] values = { 1.15m, 1.25m, 1.35m, 1.45m, 1.55m, 1.65m };
decimal sum = 0;

// Calculate true mean.
foreach (var value in values)
    sum += value;

Console.WriteLine("True mean:     {0:N2}", sum / values.Length);

// Calculate mean with rounding away from zero.
sum = 0;
foreach (var value in values)
    sum += Math.Round(value, 1, MidpointRounding.AwayFromZero);

Console.WriteLine("AwayFromZero:  {0:N2}", sum / values.Length);

// Calculate mean with rounding to nearest.
sum = 0;
foreach (var value in values)
```

```
    sum += Math.Round(value, 1, MidpointRounding.ToEven);

Console.WriteLine("ToEven:        {0:N2}", sum / values.Length);

// The example displays the following output:
//       True mean:     1.40
//       AwayFromZero:  1.45
//       ToEven:        1.40
```

By default, the Round method uses the round to nearest even convention. The following table lists the overloads of the Round method and the rounding convention that each uses.

Expand table

Overload	Rounding convention
Round(Decimal)	ToEven
Round(Double)	ToEven
Round(Decimal, Int32)	ToEven
Round(Double, Int32)	ToEven
Round(Decimal, MidpointRounding)	Determined by `mode` parameter.
Round(Double, MidpointRounding)	Determined by `mode` parameter
Round(Decimal, Int32, MidpointRounding)	Determined by `mode` parameter
Round(Double, Int32, MidpointRounding)	Determined by `mode` parameter

Rounding and precision

In order to determine whether a rounding operation involves a midpoint value, the Round method multiplies the original value to be rounded by 10^n, where *n* is the desired number of fractional digits in the return value, and then determines whether the remaining fractional portion of the value is greater than or equal to .5. This is a slight variation on a test for equality, and as discussed in the "Testing for Equality" section of the Double reference topic, tests for equality with floating-point values are problematic because of the floating-point format's issues with binary representation and precision. This means that any fractional portion of a number that is slightly less than .5 (because of a loss of precision) will not be rounded upward.

The following example illustrates the problem. It repeatedly adds .1 to 11.0 and rounds the result to the nearest integer. 11.5 should round to 12 using either of the midpoint-rounding conventions (`ToEven` or `AwayFromZero`). However, as the output from the example shows, it does not. The example uses the "R" standard numeric format string to display the floating point value's full precision, and shows that the value to be rounded has lost precision during repeated additions, and its value is actually 11.499999999999998. Because .499999999999998 is less than .5, the midpoint-rounding conventions don't come into play and the value is rounded down. As the example also shows, this problem does not occur if you assign the constant value 11.5 to a Double variable.

C#

```
public static void Example()
{
    Console.WriteLine("{0,5} {1,20:R}  {2,12} {3,15}\n",
                      "Value", "Full Precision", "ToEven",
                      "AwayFromZero");
    double value = 11.1;
    for (int ctr = 0; ctr <= 5; ctr++)
        value = RoundValueAndAdd(value);

    Console.WriteLine();

    value = 11.5;
    RoundValueAndAdd(value);
}

private static double RoundValueAndAdd(double value)
{
    Console.WriteLine("{0,5:N1} {0,20:R}  {1,12} {2,15}",
                      value, Math.Round(value, MidpointRounding.ToEven),
                      Math.Round(value, MidpointRounding.AwayFromZero));
    return value + .1;
}

// The example displays the following output:
//       Value       Full Precision        ToEven    AwayFromZero
//
//        11.1                 11.1            11              11
//        11.2                 11.2            11              11
//        11.3   11.299999999999999            11              11
//        11.4   11.399999999999999            11              11
//        11.5   11.499999999999998            11              11
//        11.6   11.599999999999998            12              12
//
//        11.5                 11.5            12              12
```

Problems of precision in rounding midpoint values are most likely to arise in the following conditions:

- When a fractional value cannot be expressed precisely in the floating-point type's binary format.
- When the value to be rounded is calculated from one or more floating-point operations.
- When the value to be rounded is a Single rather than a Double or Decimal. For more information, see the next section, Rounding and single-precision floating-point values.

In cases where the lack of precision in rounding operations is problematic, you can do the following:

- If the rounding operation calls an overload that rounds a Double value, you can change the Double to a Decimal value and call an overload that rounds a Decimal value instead. Although the Decimal data type also has problems of representation and loss of precision, these issues are far less common.
- Define a custom rounding algorithm that performs a "nearly equal" test to determine whether the value to be rounded is acceptably close to a midpoint value. The following example defines a `RoundApproximate` method that examines whether a fractional value is sufficiently near to a midpoint value to be subject to midpoint rounding. As the output from the example shows, it corrects the rounding problem shown in the previous example.

C#

```
public static void Example()
{
    Console.WriteLine("{0,5} {1,20:R}  {2,12} {3,15}\n",
                      "Value", "Full Precision", "ToEven",
                      "AwayFromZero");
    double value = 11.1;
    for (int ctr = 0; ctr <= 5; ctr++)
        value = RoundValueAndAdd(value);

    Console.WriteLine();

    value = 11.5;
    RoundValueAndAdd(value);
}

private static double RoundValueAndAdd(double value)
{
    const double tolerance = 8e-14;
```

```
    Console.WriteLine("{0,5:N1} {0,20:R}  {1,12} {2,15}",
                      value,
                      RoundApproximate(value, 0, tolerance,
MidpointRounding.ToEven),
                      RoundApproximate(value, 0, tolerance,
MidpointRounding.AwayFromZero));
    return value + .1;
}

private static double RoundApproximate(double dbl, int digits, double
margin,
                                   MidpointRounding mode)
{
    double fraction = dbl * Math.Pow(10, digits);
    double value = Math.Truncate(fraction);
    fraction = fraction - value;
    if (fraction == 0)
        return dbl;

    double tolerance = margin * dbl;
    // Determine whether this is a midpoint value.
    if ((fraction >= .5 - tolerance) & (fraction <= .5 + tolerance))
    {
        if (mode == MidpointRounding.AwayFromZero)
            return (value + 1) / Math.Pow(10, digits);
        else
           if (value % 2 != 0)
            return (value + 1) / Math.Pow(10, digits);
        else
            return value / Math.Pow(10, digits);
    }
    // Any remaining fractional value greater than .5 is not a midpoint
value.
    if (fraction > .5)
        return (value + 1) / Math.Pow(10, digits);
    else
        return value / Math.Pow(10, digits);
}

// The example displays the following output:
//       Value       Full Precision        ToEven    AwayFromZero
//
//        11.1                 11.1            11              11
//        11.2                 11.2            11              11
//        11.3   11.299999999999999            11              11
//        11.4   11.399999999999999            11              11
//        11.5   11.499999999999998            12              12
//        11.6   11.599999999999998            12              12
//
//        11.5                 11.5            12              12
```

Rounding and single-precision floating-point values

The Round method includes overloads that accept arguments of type Decimal and Double. There are no methods that round values of type Single. If you pass a Single value to one of the overloads of the Round method, it is cast (in C#) or converted (in Visual Basic) to a Double, and the corresponding Round overload with a Double parameter is called. Although this is a widening conversion, it often involves a loss of precision, as the following example illustrates. When a Single value of 16.325 is passed to the Round method and rounded to two decimal places using the rounding to nearest convention, the result is 16.33 and not the expected result of 16.32.

C#

```
Single value = 16.325f;
Console.WriteLine("Widening Conversion of {0:R} (type {1}) to {2:R} (type
{3}): ",
                  value, value.GetType().Name, (double)value,
                  ((double)(value)).GetType().Name);
Console.WriteLine(Math.Round(value, 2));
Console.WriteLine(Math.Round(value, 2, MidpointRounding.AwayFromZero));
Console.WriteLine();

Decimal decValue = (decimal)value;
Console.WriteLine("Cast of {0:R} (type {1}) to {2} (type {3}): ",
                  value, value.GetType().Name, decValue,
                  decValue.GetType().Name);
Console.WriteLine(Math.Round(decValue, 2));
Console.WriteLine(Math.Round(decValue, 2, MidpointRounding.AwayFromZero));

// The example displays the following output:
//    Widening Conversion of 16.325 (type Single) to 16.325000762939453
(type Double):
//    16.33
//    16.33
//
//    Cast of 16.325 (type Single) to 16.325 (type Decimal):
//    16.32
//    16.33
```

This unexpected result is due to a loss of precision in the conversion of the Single value to a Double. Because the resulting Double value of 16.325000762939453 is not a midpoint value and is greater than 16.325, it is always rounded upward.

In many cases, as the example illustrates, the loss of precision can be minimized or eliminated by casting or converting the Single value to a Decimal. Note that, because this is a narrowing conversion, it requires using a cast operator or calling a conversion method.

Round(Double, Int32, MidpointRounding)

Rounds a double-precision floating-point value to a specified number of fractional digits using the specified rounding convention.

```
C#

public static double Round (double value, int digits, MidpointRounding
mode);
```

Parameters

`value` Double

A double-precision floating-point number to be rounded.

`digits` Int32

The number of fractional digits in the return value.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Double

The number that has `digits` fractional digits that `value` is rounded to. If `value` has fewer fractional digits than `digits`, `value` is returned unchanged.

Exceptions

ArgumentOutOfRangeException

`digits` is less than 0 or greater than 15.

ArgumentException

`mode` is not a valid value of MidpointRounding.

Remarks

The value of the `digits` argument can range from 0 to 15. The maximum number of integral and fractional digits supported by the Double type is 15.

See Midpoint values and rounding conventions for information about rounding numbers with midpoint values.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `value` argument is Double.NaN, the method returns Double.NaN. If `value` is Double.PositiveInfinity or Double.NegativeInfinity, the method returns Double.PositiveInfinity or Double.NegativeInfinity, respectively.

Example

The following example demonstrates how to use the Round(Double, Int32, MidpointRounding) method with the MidpointRounding enumeration.

C#

```
// Round a positive and a negative value using the default.
double result = Math.Round(3.45, 1);
Console.WriteLine($"{result,4} = Math.Round({3.45,5}, 1)");
result = Math.Round(-3.45, 1);
Console.WriteLine($"{result,4} = Math.Round({-3.45,5}, 1)\n");

// Round a positive value using a MidpointRounding value.
result = Math.Round(3.45, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result,4} = Math.Round({3.45,5}, 1,
MidpointRounding.ToEven)");
result = Math.Round(3.45, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result,4} = Math.Round({3.45,5}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(3.47, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result,4} = Math.Round({3.47,5}, 1,
MidpointRounding.ToZero)\n");

// Round a negative value using a MidpointRounding value.
result = Math.Round(-3.45, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result,4} = Math.Round({-3.45,5}, 1,
MidpointRounding.ToEven)");
result = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result,4} = Math.Round({-3.45,5}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(-3.47, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result,4} = Math.Round({-3.47,5}, 1,
MidpointRounding.ToZero)\n");

// The example displays the following output:
```

```
//          3.4 = Math.Round( 3.45, 1)
//         -3.4 = Math.Round(-3.45, 1)

//          3.4 = Math.Round(3.45, 1, MidpointRounding.ToEven)
//          3.5 = Math.Round(3.45, 1, MidpointRounding.AwayFromZero)
//          3.4 = Math.Round(3.47, 1, MidpointRounding.ToZero)

//         -3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
//         -3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
//         -3.4 = Math.Round(-3.47, 1, MidpointRounding.ToZero)
```

Notes to Callers

Because of the loss of precision that can result from representing decimal values as floating-point numbers or performing arithmetic operations on floating-point values, in some cases the Round(Double, Int32, MidpointRounding) method may not appear to round midpoint values as specified by the `mode` parameter. This is illustrated in the following example, where 2.135 is rounded to 2.13 instead of 2.14. This occurs because internally the method multiplies `value` by 10^{digits}, and the multiplication operation in this case suffers from a loss of precision.

C#

```
double[] values = { 2.125, 2.135, 2.145, 3.125, 3.135, 3.145 };
foreach (double value in values)
   Console.WriteLine("{0} --> {1}", value,
                     Math.Round(value, 2,
MidpointRounding.AwayFromZero));

// The example displays the following output:
//       2.125 --> 2.13
//       2.135 --> 2.13
//       2.145 --> 2.15
//       3.125 --> 3.13
//       3.135 --> 3.14
//       3.145 --> 3.15
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Decimal, Int32, MidpointRounding)

Rounds a decimal value to a specified number of fractional digits using the specified rounding convention.

C#

```
public static decimal Round (decimal d, int decimals, MidpointRounding
mode);
```

Parameters

`d` Decimal

A decimal number to be rounded.

`decimals` Int32

The number of decimal places in the return value.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Decimal

The number with `decimals` fractional digits that `d` is rounded to. If `d` has fewer fractional digits than `decimals`, `d` is returned unchanged.

Exceptions

ArgumentOutOfRangeException

`decimals` is less than 0 or greater than 28.

ArgumentException

`mode` is not a valid value of MidpointRounding.

OverflowException

The result is outside the range of a Decimal.

Remarks

See Midpoint values and rounding conventions for information about rounding numbers with midpoint values.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

The value of the `decimals` argument can range from 0 to 28.

Example

The following example demonstrates how to use the Round method with the MidpointRounding enumeration.

C#

```
decimal result;

// Round a positive value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
MidpointRounding.ToEven)");
result = Math.Round(3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(3.47m, 1, MidpointRounding.ToZero);
```

```
Console.WriteLine($"{result} = Math.Round({3.47m}, 1,
MidpointRounding.ToZero)\n");

// Round a negative value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(-3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.ToEven)");
result = Math.Round(-3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(-3.47m, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result} = Math.Round({-3.47m}, 1,
MidpointRounding.ToZero)\n");

/*
This code example produces the following results:

3.4 = Math.Round(3.45, 1, MidpointRounding.ToEven)
3.5 = Math.Round(3.45, 1, MidpointRounding.AwayFromZero)
3.4 = Math.Round(3.47, 1, MidpointRounding.ToZero)

-3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
-3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
-3.4 = Math.Round(-3.47, 1, MidpointRounding.ToZero)
*/
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Double, MidpointRounding)

Rounds a double-precision floating-point value to an integer using the specified rounding convention.

```
C#

public static double Round (double value, MidpointRounding mode);
```

Parameters

`value` Double

A double-precision floating-point number to be rounded.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Double

The integer that `value` is rounded to. This method returns a Double instead of an integral type.

Exceptions

ArgumentException

`mode` is not a valid value of MidpointRounding.

Remarks

See Midpoint values and rounding conventions for information about rounding numbers with midpoint values.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `value` argument is Double.NaN, the method returns Double.NaN. If `value` is Double.PositiveInfinity or Double.NegativeInfinity, the method returns Double.PositiveInfinity or Double.NegativeInfinity, respectively.

Example

The following example displays values returned by the Round(Double, MidpointRounding) method with different `mode` values.

C#

```
Double[] values = { 12.0, 12.1, 12.2, 12.3, 12.4, 12.5, 12.6,
                    12.7, 12.8, 12.9, 13.0 };
Console.WriteLine($"{"Value",-10} {"Default",-10} {"ToEven",-10}
{"AwayFromZero",-15} {"ToZero",-15}");

foreach (var value in values)
    Console.WriteLine($"{value,-10:R} {Math.Round(value),-10} " +
        $"{Math.Round(value, MidpointRounding.ToEven),-10} " +
        $"{Math.Round(value, MidpointRounding.AwayFromZero),-15} " +
        $"{Math.Round(value, MidpointRounding.ToZero),-15}");

// The example displays the following output:
//       Value      Default    ToEven     AwayFromZero    ToZero
//       12         12         12         12              12
//       12.1       12         12         12              12
//       12.2       12         12         12              12
//       12.3       12         12         12              12
//       12.4       12         12         12              12
//       12.5       12         12         13              12
//       12.6       13         13         13              12
//       12.7       13         13         13              12
//       12.8       13         13         13              12
//       12.9       13         13         13              12
//       13         13         13         13              13
```

Notes to Callers

Because of the loss of precision that can result from representing decimal values as floating-point numbers or performing arithmetic operations on floating-point values, in some cases the Round(Double, MidpointRounding) method may not appear to round midpoint values to the nearest even integer. In the following example, because the floating-point value .1 has no finite binary representation, the first call to the Round(Double) method with a value of 11.5 returns 11 instead of 12.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double value = 11.1;
      for (int ctr = 0; ctr <= 5; ctr++)
         value = RoundValueAndAdd(value);

      Console.WriteLine();

      value = 11.5;
      RoundValueAndAdd(value);
   }

   private static double RoundValueAndAdd(double value)
   {
      Console.WriteLine("{0} --> {1}", value, Math.Round(value,
                        MidpointRounding.AwayFromZero));
      return value + .1;
   }
}
// The example displays the following output:
//       11.1 --> 11
//       11.2 --> 11
//       11.3 --> 11
//       11.4 --> 11
//       11.5 --> 11
//       11.6 --> 12
//
//       11.5 --> 12
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Double, Int32)

Rounds a double-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.

```
C#
public static double Round (double value, int digits);
```

Parameters

`value` Double

A double-precision floating-point number to be rounded.

`digits` Int32

The number of fractional digits in the return value.

Returns

Double

The number nearest to `value` that contains a number of fractional digits equal to `digits`.

Exceptions

ArgumentOutOfRangeException

`digits` is less than 0 or greater than 15.

Remarks

The value of the `digits` argument can range from 0 to 15. The maximum number of integral and fractional digits supported by the Double type is 15.

This method uses the default rounding convention of MidpointRounding.ToEven. See Midpoint values and rounding conventions for information about rounding numbers with midpoint values.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `value` argument is Double.NaN, the method returns Double.NaN. If `value` is Double.PositiveInfinity or Double.NegativeInfinity, the method returns Double.PositiveInfinity or Double.NegativeInfinity, respectively.

Example

The following example rounds double values with two fractional digits to doubles that have a single fractional digit.

C#

```
Math.Round(3.44, 1); //Returns 3.4.
Math.Round(3.45, 1); //Returns 3.4.
Math.Round(3.46, 1); //Returns 3.5.

Math.Round(4.34, 1); // Returns 4.3
Math.Round(4.35, 1); // Returns 4.4
Math.Round(4.36, 1); // Returns 4.4
```

Notes to Callers

Because of the loss of precision that can result from representing decimal values as floating-point numbers or performing arithmetic operations on floating-point values, in some cases the Round(Double, Int32) method may not appear to round midpoint values to the nearest even value in the `digits` decimal position. This is illustrated in the following example, where 2.135 is rounded to 2.13 instead of 2.14. This occurs because internally the method multiplies `value` by 10^{digits}, and the multiplication operation in this case suffers from a loss of precision.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double[] values = { 2.125, 2.135, 2.145, 3.125, 3.135, 3.145 };
      foreach (double value in values)
         Console.WriteLine("{0} --> {1}", value, Math.Round(value, 2));
   }
}
// The example displays the following output:
//       2.125 --> 2.12
//       2.135 --> 2.13
//       2.145 --> 2.14
//       3.125 --> 3.12
//       3.135 --> 3.14
//       3.145 --> 3.14
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Decimal, Int32)

Rounds a decimal value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.

```
C#
public static decimal Round (decimal d, int decimals);
```

Parameters

d Decimal

A decimal number to be rounded.

decimals Int32

The number of decimal places in the return value.

Returns

Decimal

The number nearest to `d` that contains a number of fractional digits equal to `decimals`.

Exceptions

ArgumentOutOfRangeException

`decimals` is less than 0 or greater than 28.

OverflowException

The result is outside the range of a Decimal.

Remarks

The value of the `decimals` argument can range from 0 to 28.

This method uses the default rounding convention of MidpointRounding.ToEven. For information about rounding numbers with midpoint values, see Midpoint values and rounding conventions.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

Example

The following example rounds decimal values with two fractional digits to values that have a single fractional digit.

C#

```
Console.WriteLine(Math.Round(3.44m, 1));
Console.WriteLine(Math.Round(3.45m, 1));
Console.WriteLine(Math.Round(3.46m, 1));
Console.WriteLine();

Console.WriteLine(Math.Round(4.34m, 1));
Console.WriteLine(Math.Round(4.35m, 1));
Console.WriteLine(Math.Round(4.36m, 1));

// The example displays the following output:
//       3.4
//       3.4
//       3.5
//
//       4.3
//       4.4
//       4.4
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Double)

Rounds a double-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.

```
C#

public static double Round (double a);
```

Parameters

a Double

A double-precision floating-point number to be rounded.

Returns

Double

The integer nearest `a`. If the fractional component of `a` is halfway between two integers, one of which is even and the other odd, then the even number is returned. Note that this method returns a Double instead of an integral type.

Remarks

This method uses the default rounding convention of MidpointRounding.ToEven. For information about rounding numbers with midpoint values, see Midpoint values and rounding conventions.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `a` argument is Double.NaN, the method returns Double.NaN. If `a` is Double.PositiveInfinity or Double.NegativeInfinity, the method returns Double.PositiveInfinity or Double.NegativeInfinity, respectively.

Starting with Visual Basic 15.8, the performance of Double-to-integer conversion is optimized if you pass the value returned by the `Round` method to the any of the

integral conversion functions, or if the Double value returned by `Round` is automatically converted to an integer with Option Strict set to Off. This optimization allows code to run faster -- up to twice as fast for code that does a large number of conversions to integer types. The following example illustrates such optimized conversions:

VB

```
Dim d1 As Double = 1043.75133
Dim i1 As Integer = CInt(Math.Ceiling(d1))        ' Result: 1044

Dim d2 As Double = 7968.4136
Dim i2 As Integer = CInt(Math.Ceiling(d2))        ' Result: 7968
```

Example

The following example demonstrates rounding to the nearest integer value.

C#

```
Console.WriteLine("Classic Math.Round in CSharp");
Console.WriteLine(Math.Round(4.4)); // 4
Console.WriteLine(Math.Round(4.5)); // 4
Console.WriteLine(Math.Round(4.6)); // 5
Console.WriteLine(Math.Round(5.5)); // 6
```

Notes to Callers

Because of the loss of precision that can result from representing decimal values as floating-point numbers or performing arithmetic operations on floating-point values, in some cases the Round(Double) method may not appear to round midpoint values to the nearest even integer. In the following example, because the floating-point value .1 has no finite binary representation, the first call to the Round(Double) method with a value of 11.5 returns 11 instead of 12.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      double value = 11.1;
      for (int ctr = 0; ctr <= 5; ctr++)
```

```
            value = RoundValueAndAdd(value);

        Console.WriteLine();

        value = 11.5;
        RoundValueAndAdd(value);
    }

    private static double RoundValueAndAdd(double value)
    {
        Console.WriteLine("{0} --> {1}", value, Math.Round(value));
        return value + .1;
    }
}
// The example displays the following output:
//       11.1 --> 11
//       11.2 --> 11
//       11.3 --> 11
//       11.4 --> 11
//       11.5 --> 11
//       11.6 --> 12
//
//       11.5 --> 12
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Decimal)

Rounds a decimal value to the nearest integral value, and rounds midpoint values to the nearest even number.

C#

```
public static decimal Round (decimal d);
```

Parameters

d Decimal

A decimal number to be rounded.

Returns

Decimal

The integer nearest the `d` parameter. If the fractional component of `d` is halfway between two integers, one of which is even and the other odd, the even number is returned. Note that this method returns a Decimal instead of an integral type.

Exceptions

OverflowException

The result is outside the range of a Decimal.

Examples

The following example demonstrates the Round(Decimal) method. The Decimal value of 4.5 rounds to 4 rather than 5, because this overload uses the default ToEven convention.

C#

```
for (decimal value = 4.2m; value <= 4.8m; value+=.1m )
   Console.WriteLine("{0} --> {1}", value, Math.Round(value));
// The example displays the following output:
//       4.2 --> 4
//       4.3 --> 4
//       4.4 --> 4
//       4.5 --> 4
//       4.6 --> 5
//       4.7 --> 5
//       4.8 --> 5
```

Remarks

This method uses the default rounding convention of MidpointRounding.ToEven. For information about rounding numbers with midpoint values, see Midpoint values and rounding conventions.

> ⓘ **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Round(Decimal, MidpointRounding)

Rounds a decimal value an integer using the specified rounding convention.

```
C#
```

```
public static decimal Round (decimal d, MidpointRounding mode);
```

Parameters

`d` Decimal

A decimal number to be rounded.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Decimal

The integer that `d` is rounded to. This method returns a Decimal instead of an integral type.

Exceptions

ArgumentException

`mode` is not a valid value of MidpointRounding.

OverflowException

The result is outside the range of a Decimal.

Remarks

For information about rounding numbers with midpoint values, see Midpoint values and rounding conventions.

> ⓘ **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

Example

The following example displays values returned by the Round(Decimal, MidpointRounding) method with different `mode` values.

C#

```
Console.WriteLine($"{"Value",-10} {"Default",-10} {"ToEven",-10}
{"AwayFromZero",-15} {"ToZero",-15}");
for (decimal value = 12.0m; value <= 13.0m; value += 0.1m)
    Console.WriteLine($"{value,-10} {Math.Round(value),-10} " +
        $"{Math.Round(value, MidpointRounding.ToEven),-10} " +
        $"{Math.Round(value, MidpointRounding.AwayFromZero),-15} " +
        $"{Math.Round(value, MidpointRounding.ToZero),-15}");

// The example displays the following output:
//       Value      Default    ToEven     AwayFromZero    ToZero
//       12.0       12         12         12              12
//       12.1       12         12         12              12
//       12.2       12         12         12              12
//       12.3       12         12         12              12
//       12.4       12         12         12              12
//       12.5       12         12         13              12
//       12.6       13         13         13              12
//       12.7       13         13         13              12
//       12.8       13         13         13              12
//       12.9       13         13         13              12
//       13.0       13         13         13              13
```

See also

- Ceiling(Double)
- Floor(Double)
- Round(Decimal, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.ScaleB(Double, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns x * 2^n computed efficiently.

```
C#
public static double ScaleB (double x, int n);
```

Parameters

x Double

A double-precision floating-point number that specifies the base value.

n Int32

A 32-bit integer that specifies the power.

Returns

Double

x * 2^n computed efficiently.

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Sign Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an integer that indicates the sign of a number.

Overloads

Expand table

Sign(IntPtr)	Returns an integer that indicates the sign of a native sized signed integer.
Sign(Single)	Returns an integer that indicates the sign of a single-precision floating-point number.
Sign(SByte)	Returns an integer that indicates the sign of an 8-bit signed integer.
Sign(Int64)	Returns an integer that indicates the sign of a 64-bit signed integer.
Sign(Decimal)	Returns an integer that indicates the sign of a decimal number.
Sign(Int16)	Returns an integer that indicates the sign of a 16-bit signed integer.
Sign(Double)	Returns an integer that indicates the sign of a double-precision floating-point number.
Sign(Int32)	Returns an integer that indicates the sign of a 32-bit signed integer.

Sign(IntPtr)

Returns an integer that indicates the sign of a native sized signed integer.

C#

```
public static int Sign (IntPtr value);
```

Parameters

`value` nint

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Examples

The following example demonstrates how to use the Sign(IntPtr) method to determine the sign of an IntPtr value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
```

```
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Sign(Single)

Returns an integer that indicates the sign of a single-precision floating-point number.

```
C#
public static int Sign (float value);
```

Parameters

`value` Single

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Exceptions

ArithmeticException

`value` is equal to NaN.

Examples

The following example demonstrates how to use the Sign(Single) method to determine the sign of a Single value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
```

```
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sign(SByte)

> Important
>
> This API is not CLS-compliant.

Returns an integer that indicates the sign of an 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public static int Sign (sbyte value);
```

Parameters

`value` SByte

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Attributes CLSCompliantAttribute

Examples

The following example demonstrates how to use the Sign(SByte) method to determine the sign of an SByte value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
```

```
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
```

```
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sign(Int64)

Returns an integer that indicates the sign of a 64-bit signed integer.

C#

```
public static int Sign (long value);
```

Parameters

`value` Int64

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Examples

The following example demonstrates how to use the Sign(Int64) method to determine the sign of an Int64 value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
```

```
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sign(Decimal)

Returns an integer that indicates the sign of a decimal number.

C#

```
public static int Sign (decimal value);
```

Parameters

`value` Decimal

A signed decimal number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Examples

The following example demonstrates how to use the Sign(Decimal) method to determine the sign of a Decimal value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
```

```
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
```

```
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sign(Int16)

Returns an integer that indicates the sign of a 16-bit signed integer.

C#

```
public static int Sign (short value);
```

Parameters

`value` Int16

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Examples

The following example demonstrates how to use the Sign(Int16) method to determine the sign of an Int16 value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
```

```
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Sign(Double)

Returns an integer that indicates the sign of a double-precision floating-point number.

```
C#
public static int Sign (double value);
```

Parameters

`value` Double

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Exceptions

ArithmeticException

`value` is equal to NaN.

Examples

The following example demonstrates how to use the Sign(Double) method to determine the sign of a Double value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;

        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
```

```
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Sign(Int32)

Returns an integer that indicates the sign of a 32-bit signed integer.

C#

```
public static int Sign (int value);
```

Parameters

`value` Int32

A signed number.

Returns

Int32

A number that indicates the sign of `value`, as shown in the following table.

Expand table

Return value	Meaning
-1	`value` is less than zero.
0	`value` is equal to zero.
1	`value` is greater than zero.

Examples

The following example demonstrates how to use the Sign(Int32) method to determine the sign of an Int32 value and display it to the console.

C#

```
// This example demonstrates Math.Sign()
using System;

class Sample
{
    public static void Main()
    {
        string str = "{0}: {1,3} is {2} zero.";
        string nl = Environment.NewLine;
        byte     xByte1    = 0;
        short    xShort1   = -2;
        int      xInt1     = -3;
        long     xLong1    = -4;
        float    xSingle1  = 0.0f;
        double   xDouble1  = 6.0;
        Decimal  xDecimal1 = -7m;
        nint     xIntPtr1  = 8;

        // The following type is not CLS-compliant.
        sbyte    xSbyte1   = -101;
```

```
        Console.WriteLine($"{nl}Test the sign of the following types of
values:");
        Console.WriteLine(str, "Byte   ", xByte1,
Test(Math.Sign(xByte1)));
        Console.WriteLine(str, "Int16  ", xShort1,
Test(Math.Sign(xShort1)));
        Console.WriteLine(str, "Int32  ", xInt1, Test(Math.Sign(xInt1)));
        Console.WriteLine(str, "Int64  ", xLong1,
Test(Math.Sign(xLong1)));
        Console.WriteLine(str, "Single ", xSingle1,
Test(Math.Sign(xSingle1)));
        Console.WriteLine(str, "Double ", xDouble1,
Test(Math.Sign(xDouble1)));
        Console.WriteLine(str, "Decimal", xDecimal1,
Test(Math.Sign(xDecimal1)));
        Console.WriteLine(str, "IntPtr", xIntPtr1,
Test(Math.Sign(xIntPtr1)));

        Console.WriteLine($"{nl}The following type is not CLS-
compliant.");
        Console.WriteLine(str, "SByte  ", xSbyte1,
Test(Math.Sign(xSbyte1)));
    }

    public static string Test(int compare)
    {
        if (compare == 0)
            return "equal to";
        else if (compare < 0)
            return "less than";
        else
            return "greater than";
    }
}
/*
This example produces the following results:

Test the sign of the following types of values:
Byte   :   0 is equal to zero.
Int16  :  -2 is less than zero.
Int32  :  -3 is less than zero.
Int64  :  -4 is less than zero.
Single :   0 is equal to zero.
Double :   6 is greater than zero.
Decimal:  -7 is less than zero.
IntPtr:    8 is greater than zero.

The following type is not CLS-compliant.
SByte  : -101 is less than zero.
*/
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Sin(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the sine of the specified angle.

C#

```
public static double Sin (double a);
```

Parameters

a Double

An angle, measured in radians.

Returns

Double

The sine of `a`. If `a` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Examples

The following example uses Sin to evaluate certain trigonometric identities for selected angles.

C#

```
// Example for the trigonometric Math.Sin( double )
// and Math.Cos( double ) methods.
using System;

class SinCos
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of trigonometric " +
```

```
            "Math.Sin( double ), Math.Cos( double ), and Math.SinCos( double
)\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Convert selected values for X to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin^2(X) + cos^2(X) == 1\n" +
                           "   sin(2 * X) == 2 * sin(X) * cos(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );

        UseSineCosine(15.0);
        UseSineCosine(30.0);
        UseSineCosine(45.0);

        Console.WriteLine(
            "\nConvert selected values for X and Y to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin(X + Y) == sin(X) * cos(Y) + cos(X) *
sin(Y)" );
        Console.WriteLine( "   cos(X + Y) == cos(X) * cos(Y) - sin(X) *
sin(Y)" );

        UseTwoAngles(15.0, 30.0);
        UseTwoAngles(30.0, 45.0);

        Console.WriteLine(
            "\nWhen you have calls to sin(X) and cos(X) they \n" +
            "can be replaced with a single call to sincos(x):" );

        UseCombinedSineCosine(15.0);
        UseCombinedSineCosine(30.0);
        UseCombinedSineCosine(45.0);
    }

    // Evaluate trigonometric identities with a given angle.
    static void UseCombinedSineCosine(double degrees)
    {
        double angle = Math.PI * degrees / 180.0;
        (double sinAngle, double cosAngle) = Math.SinCos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                           Math.SinCos({0} deg) == ({1:E16},
{2:E16})",
            degrees, sinAngle, cosAngle);
        Console.WriteLine(
            "(double sin, double cos) = Math.SinCos({0} deg)",
            degrees );
        Console.WriteLine(
            "sin^2 + cos^2 == {0:E16}",
            sinAngle * sinAngle + cosAngle * cosAngle );
    }

    // Evaluate trigonometric identities with a given angle.
```

```
    static void UseSineCosine(double degrees)
    {
        double angle    = Math.PI * degrees / 180.0;
        double sinAngle = Math.Sin(angle);
        double cosAngle = Math.Cos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                            Math.Sin({0} deg) == {1:E16}\n" +
            "                            Math.Cos({0} deg) == {2:E16}",
            degrees, Math.Sin(angle), Math.Cos(angle) );
        Console.WriteLine(
            "(Math.Sin({0} deg))^2 + (Math.Cos({0} deg))^2 == {1:E16}",
            degrees, sinAngle * sinAngle + cosAngle * cosAngle );

        // Evaluate sin(2 * X) == 2 * sin(X) * cos(X).
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            2.0 * degrees, Math.Sin(2.0 * angle) );
        Console.WriteLine(
            "    2 * Math.Sin({0} deg) * Math.Cos({0} deg) == {1:E16}",
            degrees, 2.0 * sinAngle * cosAngle );

        // Evaluate cos(2 * X) == cos^2(X) - sin^2(X).
        Console.WriteLine(
            "                            Math.Cos({0} deg) == {1:E16}",
            2.0 * degrees, Math.Cos(2.0 * angle) );
        Console.WriteLine(
            "(Math.Cos({0} deg))^2 - (Math.Sin({0} deg))^2 == {1:E16}",
            degrees, cosAngle * cosAngle - sinAngle * sinAngle );
    }

    // Evaluate trigonometric identities that are functions of two angles.
    static void UseTwoAngles(double degreesX, double degreesY)
    {
        double  angleX  = Math.PI * degreesX / 180.0;
        double  angleY  = Math.PI * degreesY / 180.0;

        // Evaluate sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y).
        Console.WriteLine(
            "\n        Math.Sin({0} deg) * Math.Cos({1} deg) +\n" +
            "        Math.Cos({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Sin(angleX) * Math.Cos(angleY) +
            Math.Cos(angleX) * Math.Sin(angleY));
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Sin(angleX + angleY));

        // Evaluate cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y).
        Console.WriteLine(
            "        Math.Cos({0} deg) * Math.Cos({1} deg) -\n" +
            "        Math.Sin({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Cos(angleX) * Math.Cos(angleY) -
            Math.Sin(angleX) * Math.Sin(angleY));
        Console.WriteLine(
```

```
            "                                Math.Cos({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Cos(angleX + angleY));
    }
}

/*
This example of trigonometric Math.Sin( double ) and Math.Cos( double )
generates the following output.

Convert selected values for X to radians
and evaluate these trigonometric identities:
   sin^2(X) + cos^2(X) == 1
   sin(2 * X) == 2 * sin(X) * cos(X)
   cos(2 * X) == cos^2(X) - sin^2(X)

                            Math.Sin(15 deg) == 2.5881904510252074E-001
                            Math.Cos(15 deg) == 9.6592582628906831E-001
(Math.Sin(15 deg))^2 + (Math.Cos(15 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(30 deg) == 4.9999999999999994E-001
    2 * Math.Sin(15 deg) * Math.Cos(15 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Cos(15 deg))^2 - (Math.Sin(15 deg))^2 == 8.6602540378443871E-001

                            Math.Sin(30 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Sin(30 deg))^2 + (Math.Cos(30 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(60 deg) == 8.6602540378443860E-001
    2 * Math.Sin(30 deg) * Math.Cos(30 deg) == 8.6602540378443860E-001
                            Math.Cos(60 deg) == 5.0000000000000011E-001
(Math.Cos(30 deg))^2 - (Math.Sin(30 deg))^2 == 5.0000000000000022E-001

                            Math.Sin(45 deg) == 7.0710678118654746E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001
(Math.Sin(45 deg))^2 + (Math.Cos(45 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(90 deg) == 1.0000000000000000E+000
    2 * Math.Sin(45 deg) * Math.Cos(45 deg) == 1.0000000000000000E+000
                            Math.Cos(90 deg) == 6.1230317691118863E-017
(Math.Cos(45 deg))^2 - (Math.Sin(45 deg))^2 == 2.2204460492503131E-016

Convert selected values for X and Y to radians
and evaluate these trigonometric identities:
   sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y)
   cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y)

        Math.Sin(15 deg) * Math.Cos(30 deg) +
        Math.Cos(15 deg) * Math.Sin(30 deg) == 7.0710678118654746E-001
                            Math.Sin(45 deg) == 7.0710678118654746E-001
        Math.Cos(15 deg) * Math.Cos(30 deg) -
        Math.Sin(15 deg) * Math.Sin(30 deg) == 7.0710678118654757E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001

        Math.Sin(30 deg) * Math.Cos(45 deg) +
        Math.Cos(30 deg) * Math.Sin(45 deg) == 9.6592582628906831E-001
                            Math.Sin(75 deg) == 9.6592582628906820E-001
        Math.Cos(30 deg) * Math.Cos(45 deg) -
```

```
        Math.Sin(30 deg) * Math.Sin(45 deg) == 2.5881904510252085E-001
                           Math.Cos(75 deg) == 2.5881904510252096E-001
*/
```

Remarks

The angle, `a`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.SinCos(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the sine and cosine of the specified angle.

C#

```
public static (double Sin, double Cos) SinCos (double x);
```

Parameters

x Double

An angle, measured in radians.

Returns

ValueTuple<Double,Double>

The sine and cosine of `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Examples

The following example uses SinCos to evaluate certain trigonometric identities for selected angles.

C#

```
// Example for the trigonometric Math.Sin( double )
// and Math.Cos( double ) methods.
using System;

class SinCos
{
    public static void Main()
    {
        Console.WriteLine(
            "This example of trigonometric " +
```

```
            "Math.Sin( double ), Math.Cos( double ), and Math.SinCos( double
)\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Convert selected values for X to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin^2(X) + cos^2(X) == 1\n" +
                           "   sin(2 * X) == 2 * sin(X) * cos(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );
        Console.WriteLine( "   cos(2 * X) == cos^2(X) - sin^2(X)" );

        UseSineCosine(15.0);
        UseSineCosine(30.0);
        UseSineCosine(45.0);

        Console.WriteLine(
            "\nConvert selected values for X and Y to radians \n" +
            "and evaluate these trigonometric identities:" );
        Console.WriteLine( "   sin(X + Y) == sin(X) * cos(Y) + cos(X) *
sin(Y)" );
        Console.WriteLine( "   cos(X + Y) == cos(X) * cos(Y) - sin(X) *
sin(Y)" );

        UseTwoAngles(15.0, 30.0);
        UseTwoAngles(30.0, 45.0);

        Console.WriteLine(
            "\nWhen you have calls to sin(X) and cos(X) they \n" +
            "can be replaced with a single call to sincos(x):" );

        UseCombinedSineCosine(15.0);
        UseCombinedSineCosine(30.0);
        UseCombinedSineCosine(45.0);
    }

    // Evaluate trigonometric identities with a given angle.
    static void UseCombinedSineCosine(double degrees)
    {
        double angle = Math.PI * degrees / 180.0;
        (double sinAngle, double cosAngle) = Math.SinCos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                           Math.SinCos({0} deg) == ({1:E16},
{2:E16})",
            degrees, sinAngle, cosAngle);
        Console.WriteLine(
            "(double sin, double cos) = Math.SinCos({0} deg)",
            degrees );
        Console.WriteLine(
            "sin^2 + cos^2 == {0:E16}",
            sinAngle * sinAngle + cosAngle * cosAngle );
    }

    // Evaluate trigonometric identities with a given angle.
```

```
    static void UseSineCosine(double degrees)
    {
        double angle    = Math.PI * degrees / 180.0;
        double sinAngle = Math.Sin(angle);
        double cosAngle = Math.Cos(angle);

        // Evaluate sin^2(X) + cos^2(X) == 1.
        Console.WriteLine(
            "\n                            Math.Sin({0} deg) == {1:E16}\n" +
            "                            Math.Cos({0} deg) == {2:E16}",
            degrees, Math.Sin(angle), Math.Cos(angle) );
        Console.WriteLine(
            "(Math.Sin({0} deg))^2 + (Math.Cos({0} deg))^2 == {1:E16}",
            degrees, sinAngle * sinAngle + cosAngle * cosAngle );

        // Evaluate sin(2 * X) == 2 * sin(X) * cos(X).
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            2.0 * degrees, Math.Sin(2.0 * angle) );
        Console.WriteLine(
            "    2 * Math.Sin({0} deg) * Math.Cos({0} deg) == {1:E16}",
            degrees, 2.0 * sinAngle * cosAngle );

        // Evaluate cos(2 * X) == cos^2(X) - sin^2(X).
        Console.WriteLine(
            "                            Math.Cos({0} deg) == {1:E16}",
            2.0 * degrees, Math.Cos(2.0 * angle) );
        Console.WriteLine(
            "(Math.Cos({0} deg))^2 - (Math.Sin({0} deg))^2 == {1:E16}",
            degrees, cosAngle * cosAngle - sinAngle * sinAngle );
    }

    // Evaluate trigonometric identities that are functions of two angles.
    static void UseTwoAngles(double degreesX, double degreesY)
    {
        double  angleX  = Math.PI * degreesX / 180.0;
        double  angleY  = Math.PI * degreesY / 180.0;

        // Evaluate sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y).
        Console.WriteLine(
            "\n        Math.Sin({0} deg) * Math.Cos({1} deg) +\n" +
            "        Math.Cos({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Sin(angleX) * Math.Cos(angleY) +
            Math.Cos(angleX) * Math.Sin(angleY));
        Console.WriteLine(
            "                            Math.Sin({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Sin(angleX + angleY));

        // Evaluate cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y).
        Console.WriteLine(
            "        Math.Cos({0} deg) * Math.Cos({1} deg) -\n" +
            "        Math.Sin({0} deg) * Math.Sin({1} deg) == {2:E16}",
            degreesX, degreesY, Math.Cos(angleX) * Math.Cos(angleY) -
            Math.Sin(angleX) * Math.Sin(angleY));
        Console.WriteLine(
```

```
            "                                Math.Cos({0} deg) == {1:E16}",
            degreesX + degreesY, Math.Cos(angleX + angleY));
    }
}

/*
This example of trigonometric Math.Sin( double ) and Math.Cos( double )
generates the following output.

Convert selected values for X to radians
and evaluate these trigonometric identities:
   sin^2(X) + cos^2(X) == 1
   sin(2 * X) == 2 * sin(X) * cos(X)
   cos(2 * X) == cos^2(X) - sin^2(X)

                            Math.Sin(15 deg) == 2.5881904510252074E-001
                            Math.Cos(15 deg) == 9.6592582628906831E-001
(Math.Sin(15 deg))^2 + (Math.Cos(15 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(30 deg) == 4.9999999999999994E-001
    2 * Math.Sin(15 deg) * Math.Cos(15 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Cos(15 deg))^2 - (Math.Sin(15 deg))^2 == 8.6602540378443871E-001

                            Math.Sin(30 deg) == 4.9999999999999994E-001
                            Math.Cos(30 deg) == 8.6602540378443871E-001
(Math.Sin(30 deg))^2 + (Math.Cos(30 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(60 deg) == 8.6602540378443860E-001
    2 * Math.Sin(30 deg) * Math.Cos(30 deg) == 8.6602540378443860E-001
                            Math.Cos(60 deg) == 5.0000000000000011E-001
(Math.Cos(30 deg))^2 - (Math.Sin(30 deg))^2 == 5.0000000000000022E-001

                            Math.Sin(45 deg) == 7.0710678118654746E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001
(Math.Sin(45 deg))^2 + (Math.Cos(45 deg))^2 == 1.0000000000000000E+000
                            Math.Sin(90 deg) == 1.0000000000000000E+000
    2 * Math.Sin(45 deg) * Math.Cos(45 deg) == 1.0000000000000000E+000
                            Math.Cos(90 deg) == 6.1230317691118863E-017
(Math.Cos(45 deg))^2 - (Math.Sin(45 deg))^2 == 2.2204460492503131E-016

Convert selected values for X and Y to radians
and evaluate these trigonometric identities:
   sin(X + Y) == sin(X) * cos(Y) + cos(X) * sin(Y)
   cos(X + Y) == cos(X) * cos(Y) - sin(X) * sin(Y)

        Math.Sin(15 deg) * Math.Cos(30 deg) +
        Math.Cos(15 deg) * Math.Sin(30 deg) == 7.0710678118654746E-001
                            Math.Sin(45 deg) == 7.0710678118654746E-001
        Math.Cos(15 deg) * Math.Cos(30 deg) -
        Math.Sin(15 deg) * Math.Sin(30 deg) == 7.0710678118654757E-001
                            Math.Cos(45 deg) == 7.0710678118654757E-001

        Math.Sin(30 deg) * Math.Cos(45 deg) +
        Math.Cos(30 deg) * Math.Sin(45 deg) == 9.6592582628906831E-001
                            Math.Sin(75 deg) == 9.6592582628906820E-001
        Math.Cos(30 deg) * Math.Cos(45 deg) -
```

```
        Math.Sin(30 deg) * Math.Sin(45 deg) == 2.5881904510252085E-001
                           Math.Cos(75 deg) == 2.5881904510252096E-001
*/
```

Remarks

The angle, `x`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Sinh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic sine of the specified angle.

C#

```
public static double Sinh (double value);
```

Parameters

`value` Double

An angle, measured in radians.

Returns

Double

The hyperbolic sine of `value`. If `value` is equal to NegativeInfinity, PositiveInfinity, or NaN, this method returns a Double equal to `value`.

Examples

The following example uses Sinh to evaluate certain hyperbolic identities for selected values.

C#

```
// Example for the hyperbolic Math.Sinh( double )
// and Math.Cosh( double ) methods.
using System;

class SinhCosh
{
    public static void Main()
    {
        Console.WriteLine(
```

```
            "This example of hyperbolic Math.Sinh( double ) " +
            "and Math.Cosh( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Evaluate these hyperbolic identities " +
            "with selected values for X:" );
        Console.WriteLine(
            "   cosh^2(X) - sinh^2(X) == 1\n" +
            "   sinh(2 * X) == 2 * sinh(X) * cosh(X)" );
        Console.WriteLine( "   cosh(2 * X) == cosh^2(X) + sinh^2(X)" );

        UseSinhCosh(0.1);
        UseSinhCosh(1.2);
        UseSinhCosh(4.9);

        Console.WriteLine(
            "\nEvaluate these hyperbolic identities " +
            "with selected values for X and Y:" );
        Console.WriteLine(
            "   sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y)" );
        Console.WriteLine(
            "   cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y)" );

        UseTwoArgs(0.1, 1.2);
        UseTwoArgs(1.2, 4.9);
    }

    // Evaluate hyperbolic identities with a given argument.
    static void UseSinhCosh(double arg)
    {
        double sinhArg = Math.Sinh(arg);
        double coshArg = Math.Cosh(arg);

        // Evaluate cosh^2(X) - sinh^2(X) == 1.
        Console.WriteLine(
            "\n                        Math.Sinh({0}) == {1:E16}\n" +
            "                        Math.Cosh({0}) == {2:E16}",
            arg, Math.Sinh(arg), Math.Cosh(arg) );
        Console.WriteLine(
            "(Math.Cosh({0}))^2 - (Math.Sinh({0}))^2 == {1:E16}",
            arg, coshArg * coshArg - sinhArg * sinhArg );

        // Evaluate sinh(2 * X) == 2 * sinh(X) * cosh(X).
        Console.WriteLine(
            "                        Math.Sinh({0}) == {1:E16}",
            2.0 * arg, Math.Sinh(2.0 * arg) );
        Console.WriteLine(
            "    2 * Math.Sinh({0}) * Math.Cosh({0}) == {1:E16}",
            arg, 2.0 * sinhArg * coshArg );

        // Evaluate cosh(2 * X) == cosh^2(X) + sinh^2(X).
        Console.WriteLine(
            "                        Math.Cosh({0}) == {1:E16}",
            2.0 * arg, Math.Cosh(2.0 * arg) );
        Console.WriteLine(
```

```
            "(Math.Cosh({0}))^2 + (Math.Sinh({0}))^2 == {1:E16}",
            arg, coshArg * coshArg + sinhArg * sinhArg );
    }

    // Evaluate hyperbolic identities that are functions of two arguments.
    static void UseTwoArgs(double argX, double argY)
    {
        // Evaluate sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y).
        Console.WriteLine(
            "\n        Math.Sinh({0}) * Math.Cosh({1}) +\n" +
            "        Math.Cosh({0}) * Math.Sinh({1}) == {2:E16}",
            argX, argY, Math.Sinh(argX) * Math.Cosh(argY) +
            Math.Cosh(argX) * Math.Sinh(argY));
        Console.WriteLine(
            "                        Math.Sinh({0}) == {1:E16}",
            argX + argY, Math.Sinh(argX + argY));

        // Evaluate cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y).
        Console.WriteLine(
            "        Math.Cosh({0}) * Math.Cosh({1}) +\n" +
            "        Math.Sinh({0}) * Math.Sinh({1}) == {2:E16}",
            argX, argY, Math.Cosh(argX) * Math.Cosh(argY) +
            Math.Sinh(argX) * Math.Sinh(argY));
        Console.WriteLine(
            "                        Math.Cosh({0}) == {1:E16}",
            argX + argY, Math.Cosh(argX + argY));
    }
}

/*
This example of hyperbolic Math.Sinh( double ) and Math.Cosh( double )
generates the following output.

Evaluate these hyperbolic identities with selected values for X:
   cosh^2(X) - sinh^2(X) == 1
   sinh(2 * X) == 2 * sinh(X) * cosh(X)
   cosh(2 * X) == cosh^2(X) + sinh^2(X)

                        Math.Sinh(0.1) == 1.0016675001984403E-001
                        Math.Cosh(0.1) == 1.0050041680558035E+000
(Math.Cosh(0.1))^2 - (Math.Sinh(0.1))^2 == 9.9999999999999989E-001
                        Math.Sinh(0.2) == 2.0133600254109399E-001
    2 * Math.Sinh(0.1) * Math.Cosh(0.1) == 2.0133600254109396E-001
                        Math.Cosh(0.2) == 1.0200667556190759E+000
(Math.Cosh(0.1))^2 + (Math.Sinh(0.1))^2 == 1.0200667556190757E+000

                        Math.Sinh(1.2) == 1.5094613554121725E+000
                        Math.Cosh(1.2) == 1.8106555673243747E+000
(Math.Cosh(1.2))^2 - (Math.Sinh(1.2))^2 == 1.0000000000000000E+000
                        Math.Sinh(2.4) == 5.4662292136760939E+000
    2 * Math.Sinh(1.2) * Math.Cosh(1.2) == 5.4662292136760939E+000
                        Math.Cosh(2.4) == 5.5569471669655064E+000
(Math.Cosh(1.2))^2 + (Math.Sinh(1.2))^2 == 5.5569471669655064E+000

                        Math.Sinh(4.9) == 6.7141166550932297E+001
```

```
                          Math.Cosh(4.9) == 6.7148613134003227E+001
(Math.Cosh(4.9))^2 - (Math.Sinh(4.9))^2 == 1.0000000000000000E+000
                          Math.Sinh(9.8) == 9.0168724361884615E+003
    2 * Math.Sinh(4.9) * Math.Cosh(4.9) == 9.0168724361884615E+003
                          Math.Cosh(9.8) == 9.0168724916400624E+003
(Math.Cosh(4.9))^2 + (Math.Sinh(4.9))^2 == 9.0168724916400606E+003

Evaluate these hyperbolic identities with selected values for X and Y:
   sinh(X + Y) == sinh(X) * cosh(Y) + cosh(X) * sinh(Y)
   cosh(X + Y) == cosh(X) * cosh(Y) + sinh(X) * sinh(Y)

        Math.Sinh(0.1) * Math.Cosh(1.2) +
        Math.Cosh(0.1) * Math.Sinh(1.2) == 1.6983824372926155E+000
                          Math.Sinh(1.3) == 1.6983824372926160E+000
        Math.Cosh(0.1) * Math.Cosh(1.2) +
        Math.Sinh(0.1) * Math.Sinh(1.2) == 1.9709142303266281E+000
                          Math.Cosh(1.3) == 1.9709142303266285E+000

        Math.Sinh(1.2) * Math.Cosh(4.9) +
        Math.Cosh(1.2) * Math.Sinh(4.9) == 2.2292776360739879E+002
                          Math.Sinh(6.1) == 2.2292776360739885E+002
        Math.Cosh(1.2) * Math.Cosh(4.9) +
        Math.Sinh(1.2) * Math.Sinh(4.9) == 2.2293000647511826E+002
                          Math.Cosh(6.1) == 2.2293000647511832E+002
*/
```

Remarks

The angle, `value`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Sqrt(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the square root of a specified number.

C#

```
public static double Sqrt (double d);
```

Parameters

`d` Double

The number whose square root is to be found.

Returns

Double

One of the values in the following table.

Expand table

`d` parameter	Return value
Zero or positive	The positive square root of `d`.
Negative	NaN
Equals NaN	NaN
Equals PositiveInfinity	PositiveInfinity

Examples

The square root of the area of a square represents the length of any side of the square. The following example displays the area of some cities in the United States and gives an impression of each city's size if it were represented by a square.

C#

```
// Create an array containing the area of some squares.
Tuple<string, double>[] areas =
               { Tuple.Create("Sitka, Alaska", 2870.3),
                 Tuple.Create("New York City", 302.6),
                 Tuple.Create("Los Angeles", 468.7),
                 Tuple.Create("Detroit", 138.8),
                 Tuple.Create("Chicago", 227.1),
                 Tuple.Create("San Diego", 325.2) };

Console.WriteLine("{0,-18} {1,14:N1} {2,30}\n", "City", "Area (mi.)",
                  "Equivalent to a square with:");

foreach (var area in areas)
  Console.WriteLine("{0,-18} {1,14:N1} {2,14:N2} miles per side",
                    area.Item1, area.Item2,
Math.Round(Math.Sqrt(area.Item2), 2));

// The example displays the following output:
//    City                   Area (mi.)   Equivalent to a square with:
//
//    Sitka, Alaska             2,870.3          53.58 miles per side
//    New York City               302.6          17.40 miles per side
//    Los Angeles                 468.7          21.65 miles per side
//    Detroit                     138.8          11.78 miles per side
//    Chicago                     227.1          15.07 miles per side
//    San Diego                   325.2          18.03 miles per side
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Pow(Double, Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Math.Tan(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the tangent of the specified angle.

C#

```
public static double Tan (double a);
```

Parameters

a Double

An angle, measured in radians.

Returns

Double

The tangent of `a`. If `a` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Examples

The following example demonstrates how to calculate the tangent of an angle and display it to the console.

C#

```
// This example demonstrates Math.Atan()
//                           Math.Atan2()
//                           Math.Tan()
using System;

class Sample
{
    public static void Main()
    {
    double x = 1.0;
```

```
    double y = 2.0;
    double angle;
    double radians;
    double result;

// Calculate the tangent of 30 degrees.
    angle = 30;
    radians = angle * (Math.PI/180);
    result = Math.Tan(radians);
    Console.WriteLine("The tangent of 30 degrees is {0}.", result);

// Calculate the arctangent of the previous tangent.
    radians = Math.Atan(result);
    angle = radians * (180/Math.PI);
    Console.WriteLine("The previous tangent is equivalent to {0} degrees.",
angle);

// Calculate the arctangent of an angle.
    String line1 = "{0}The arctangent of the angle formed by the x-axis and
";
    String line2 = "a vector to point ({0},{1}) is {2}, ";
    String line3 = "which is equivalent to {0} degrees.";

    radians = Math.Atan2(y, x);
    angle = radians * (180/Math.PI);

    Console.WriteLine(line1, Environment.NewLine);
    Console.WriteLine(line2, x, y, radians);
    Console.WriteLine(line3, angle);
    }
}
/*
This example produces the following results:

The tangent of 30 degrees is 0.577350269189626.
The previous tangent is equivalent to 30 degrees.

The arctangent of the angle formed by the x-axis and
a vector to point (1,2) is 1.10714871779409,
which is equivalent to 63.434948822922 degrees.
*/
```

Remarks

The angle, `a`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Tanh(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic tangent of the specified angle.

C#

```
public static double Tanh (double value);
```

Parameters

`value` Double

An angle, measured in radians.

Returns

Double

The hyperbolic tangent of `value`. If `value` is equal to NegativeInfinity, this method returns -1. If value is equal to PositiveInfinity, this method returns 1. If `value` is equal to NaN, this method returns NaN.

Examples

The following example uses Tanh to evaluate certain hyperbolic tangent identities for selected values.

C#

```
// Example for the hyperbolic Math.Tanh( double ) method.
using System;

class DemoTanh
{
    public static void Main()
    {
        Console.WriteLine(
```

```
            "This example of hyperbolic Math.Tanh( double )\n" +
            "generates the following output." );
        Console.WriteLine(
            "\nEvaluate these hyperbolic identities " +
            "with selected values for X:" );
        Console.WriteLine( "   tanh(X) == sinh(X) / cosh(X)" );
        Console.WriteLine(
            "   tanh(2 * X) == 2 * tanh(X) / (1 + tanh^2(X))" );

        UseTanh(0.1);
        UseTanh(1.2);
        UseTanh(4.9);

        Console.WriteLine(
            "\nEvaluate [tanh(X + Y) == (tanh(X) + tanh(Y)) " +
            "/ (1 + tanh(X) * tanh(Y))]" +
            "\nwith selected values for X and Y:" );

        UseTwoArgs(0.1, 1.2);
        UseTwoArgs(1.2, 4.9);
    }

    // Evaluate hyperbolic identities with a given argument.
    static void UseTanh(double arg)
    {
        double tanhArg = Math.Tanh(arg);

        // Evaluate tanh(X) == sinh(X) / cosh(X).
        Console.WriteLine(
            "\n                        Math.Tanh({0}) == {1:E16}\n" +
            "      Math.Sinh({0}) / Math.Cosh({0}) == {2:E16}",
            arg, tanhArg, (Math.Sinh(arg) / Math.Cosh(arg)) );

        // Evaluate tanh(2 * X) == 2 * tanh(X) / (1 + tanh^2(X)).
        Console.WriteLine(
            "                    2 * Math.Tanh({0}) /",
            arg, 2.0 * tanhArg );
        Console.WriteLine(
            "             (1 + (Math.Tanh({0}))^2) == {1:E16}",
            arg, 2.0 * tanhArg / (1.0 + tanhArg * tanhArg ) );
        Console.WriteLine(
            "                       Math.Tanh({0}) == {1:E16}",
            2.0 * arg, Math.Tanh(2.0 * arg) );
    }

    // Evaluate a hyperbolic identity that is a function of two arguments.
    static void UseTwoArgs(double argX, double argY)
    {
        // Evaluate tanh(X + Y) == (tanh(X) + tanh(Y)) / (1 + tanh(X) *
tanh(Y)).
        Console.WriteLine(
            "\n    (Math.Tanh({0}) + Math.Tanh({1})) /\n" +
            "(1 + Math.Tanh({0}) * Math.Tanh({1})) == {2:E16}",
            argX, argY, (Math.Tanh(argX) + Math.Tanh(argY)) /
            (1.0 + Math.Tanh(argX) * Math.Tanh(argY)) );
```

```
        Console.WriteLine(
            "                       Math.Tanh({0}) == {1:E16}",
            argX + argY, Math.Tanh(argX + argY));
    }
}

/*
This example of hyperbolic Math.Tanh( double )
generates the following output.

Evaluate these hyperbolic identities with selected values for X:
   tanh(X) == sinh(X) / cosh(X)
   tanh(2 * X) == 2 * tanh(X) / (1 + tanh^2(X))

                       Math.Tanh(0.1) == 9.9667994624955819E-002
      Math.Sinh(0.1) / Math.Cosh(0.1) == 9.9667994624955819E-002
                   2 * Math.Tanh(0.1) /
             (1 + (Math.Tanh(0.1))^2) == 1.9737532022490401E-001
                       Math.Tanh(0.2) == 1.9737532022490401E-001

                       Math.Tanh(1.2) == 8.3365460701215521E-001
      Math.Sinh(1.2) / Math.Cosh(1.2) == 8.3365460701215521E-001
                   2 * Math.Tanh(1.2) /
             (1 + (Math.Tanh(1.2))^2) == 9.8367485769368024E-001
                       Math.Tanh(2.4) == 9.8367485769368024E-001

                       Math.Tanh(4.9) == 9.9988910295055444E-001
      Math.Sinh(4.9) / Math.Cosh(4.9) == 9.9988910295055433E-001
                   2 * Math.Tanh(4.9) /
             (1 + (Math.Tanh(4.9))^2) == 9.9999999385024030E-001
                       Math.Tanh(9.8) == 9.9999999385024030E-001

Evaluate [tanh(X + Y) == (tanh(X) + tanh(Y)) / (1 + tanh(X) * tanh(Y))]
with selected values for X and Y:

    (Math.Tanh(0.1) + Math.Tanh(1.2)) /
(1 + Math.Tanh(0.1) * Math.Tanh(1.2)) == 8.6172315931330645E-001
                       Math.Tanh(1.3) == 8.6172315931330634E-001

    (Math.Tanh(1.2) + Math.Tanh(4.9)) /
(1 + Math.Tanh(1.2) * Math.Tanh(4.9)) == 9.9998993913939649E-001
                       Math.Tanh(6.1) == 9.9998993913939649E-001
*/
```

Remarks

The angle, `value`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Math.Truncate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the integral part of a number.

Overloads

Expand table

Truncate(Decimal)	Calculates the integral part of a specified decimal number.
Truncate(Double)	Calculates the integral part of a specified double-precision floating-point number.

Remarks

The number is rounded to the nearest integer towards zero.

Truncate(Decimal)

Calculates the integral part of a specified decimal number.

C#

```
public static decimal Truncate (decimal d);
```

Parameters

d Decimal

A number to truncate.

Returns

Decimal

The integral part of `d`; that is, the number that remains after any fractional digits have been discarded.

Examples

The following example calls the Truncate(Decimal) method to truncate both a positive and a negative Decimal value.

C#

```
decimal decimalNumber;

decimalNumber = 32.7865m;
// Displays 32
Console.WriteLine(Math.Truncate(decimalNumber));

decimalNumber = -32.9012m;
// Displays -32
Console.WriteLine(Math.Truncate(decimalNumber));
```

Remarks

Truncate rounds `d` to the nearest integer towards zero.

See also

- Round
- Ceiling(Decimal)
- Floor(Decimal)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Truncate(Double)

Calculates the integral part of a specified double-precision floating-point number.

C#

```
public static double Truncate (double d);
```

Parameters

d Double

A number to truncate.

Returns

Double

The integral part of `d`; that is, the number that remains after any fractional digits have been discarded, or one of the values listed in the following table.

Expand table

`d`	**Return value**
NaN	NaN
NegativeInfinity	NegativeInfinity
PositiveInfinity	PositiveInfinity

Examples

The following example calls the Truncate(Double) method to truncate both a positive and a negative Double value.

C#

```
double floatNumber;
```

```
floatNumber = 32.7865;
// Displays 32
Console.WriteLine(Math.Truncate(floatNumber));

floatNumber = -32.9012;
// Displays -32
Console.WriteLine(Math.Truncate(floatNumber));
```

Remarks

Truncate rounds `d` to the nearest integer towards zero.

Starting with Visual Basic 15.8, the performance of Double-to-integer conversion is optimized if you pass the value returned by the `Truncate` method to the any of the integral conversion functions, or if the Double value returned by `Truncate` is automatically converted to an integer with Option Strict set to Off. This optimization allows code to run faster -- up to twice as fast for code that does a large number of conversions to integer types. The following example illustrates such an optimized conversion:

VB

```
Dim d As Double = 164.7194
Dim i As Integer = CInt(Math.Truncate(d))     ' Result: 164
```

See also

- Round
- Ceiling(Double)
- Floor(Double)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.

C#

```
public static class MathF
```

Inheritance Object → MathF

Remarks

The static fields and methods of the `MathF` class correspond to those of the Math class, except that their parameters are of type Single rather than Double, and they return Single rather than Double values.

Fields

Expand table

E	Represents the natural logarithmic base, specified by the constant, `e`.
PI	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, p.
Tau	Represents the number of radians in one turn, specified by the constant, τ.

Methods

Expand table

Abs(Single)	Returns the absolute value of a single-precision floating-point number.
Acos(Single)	Returns the angle whose cosine is the specified number.
Acosh(Single)	Returns the angle whose hyperbolic cosine is the specified number.
Asin(Single)	Returns the angle whose sine is the specified number.
Asinh(Single)	Returns the angle whose hyperbolic sine is the specified number.
Atan(Single)	Returns the angle whose tangent is the specified number.
Atan2(Single, Single)	Returns the angle whose tangent is the quotient of two specified numbers.
Atanh(Single)	Returns the angle whose hyperbolic tangent is the specified number.
BitDecrement(Single)	Returns the largest value that compares less than a specified value.
BitIncrement(Single)	Returns the smallest value that compares greater than a specified value.
Cbrt(Single)	Returns the cube root of a specified number.
Ceiling(Single)	Returns the smallest integral value that is greater than or equal to the specified single-precision floating-point number.
CopySign(Single, Single)	Returns a value with the magnitude of `x` and the sign of `y`.
Cos(Single)	Returns the cosine of the specified angle.
Cosh(Single)	Returns the hyperbolic cosine of the specified angle.
Exp(Single)	Returns `e` raised to the specified power.
Floor(Single)	Returns the largest integral value less than or equal to the specified single-precision floating-point number.
FusedMultiplyAdd(Single, Single, Single)	Returns (x * y) + z, rounded as one ternary operation.
IEEERemainder(Single, Single)	Returns the remainder resulting from the division of a specified number by another specified number.
ILogB(Single)	Returns the base 2 integer logarithm of a specified number.
Log(Single)	Returns the natural (base `e`) logarithm of a specified number.

Log(Single, Single)	Returns the logarithm of a specified number in a specified base.
Log10(Single)	Returns the base 10 logarithm of a specified number.
Log2(Single)	Returns the base 2 logarithm of a specified number.
Max(Single, Single)	Returns the larger of two single-precision floating-point numbers.
MaxMagnitude(Single, Single)	Returns the larger magnitude of two single-precision floating-point numbers.
Min(Single, Single)	Returns the smaller of two single-precision floating-point numbers.
MinMagnitude(Single, Single)	Returns the smaller magnitude of two single-precision floating-point numbers.
Pow(Single, Single)	Returns a specified number raised to the specified power.
ReciprocalEstimate(Single)	Returns an estimate of the reciprocal of a specified number.
ReciprocalSqrt Estimate(Single)	Returns an estimate of the reciprocal square root of a specified number.
Round(Single)	Rounds a single-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Single, Int32)	Rounds a single-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.
Round(Single, Int32, Midpoint Rounding)	Rounds a single-precision floating-point value to a specified number of fractional digits using the specified rounding convention.
Round(Single, Midpoint Rounding)	Rounds a single-precision floating-point value to an integer using the specified rounding convention.
ScaleB(Single, Int32)	Returns x * 2^n computed efficiently.
Sign(Single)	Returns an integer that indicates the sign of a single-precision floating-point number.
Sin(Single)	Returns the sine of the specified angle.
SinCos(Single)	Returns the sine and cosine of the specified angle.
Sinh(Single)	Returns the hyperbolic sine of the specified angle.
Sqrt(Single)	Returns the square root of a specified number.

Tan(Single)	Returns the tangent of the specified angle.
Tanh(Single)	Returns the hyperbolic tangent of the specified angle.
Truncate(Single)	Calculates the integral part of a specified single-precision floating-point number.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.E Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the natural logarithmic base, specified by the constant, `e`.

C#

```
public const float E = 2.71828175;
```

Field Value

Value* = *2.71828175

Single

Remarks

The value of this constant is 2.71828175.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

MathF.PI Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the ratio of the circumference of a circle to its diameter, specified by the constant, p.

C#

```
public const float PI = 3.14159274;
```

Field Value

Value = 3.14159274

Single

Remarks

The value of this field is 3.14159274.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

MathF.Tau Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of radians in one turn, specified by the constant, τ.

C#

```
public const float Tau = 6.28318548;
```

Field Value

Value* = *6.28318548

Single

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Abs(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the absolute value of a single-precision floating-point number.

C#

```
public static float Abs (float x);
```

Parameters

x Single

A number that is greater than or equal to Single.MinValue, but less than or equal to Single.MaxValue.

Returns

Single

A single-precision floating-point number, x, such that 0 ≤ x ≤ Single.MaxValue.

Remarks

The absolute value of a Single is its numeric value without its sign. For example, the absolute value of both 1.2e-03 and -1.2e03 is 1.2e03.

If `x` is equal to NegativeInfinity or PositiveInfinity, the return value is PositiveInfinity. If `x` is equal to NaN, the return value is NaN.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Acos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose cosine is the specified number.

C#

```
public static float Acos (float x);
```

Parameters

x Single

A number representing a cosine, where `x` must be greater than or equal to -1, but less than or equal to 1.

Returns

Single

An angle, θ, measured in radians, such that 0 ≤ θ ≤ π.

-or-

NaN if `x` < -1 or `x` > 1 or `x` equals NaN.

Remarks

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Acosh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic cosine is the specified number.

```
C#
public static float Acosh (float x);
```

Parameters

x Single

A number representing a hyperbolic cosine, where `x` must be greater than or equal to 1, but less than or equal to PositiveInfinity.

Returns

Single

An angle, θ, measured in radians, such that 0 ≤ θ ≤ ∞.

-or-

NaN if `x` < 1 or `x` equals NaN.

Remarks

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Asin(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose sine is the specified number.

C#

```
public static float Asin (float x);
```

Parameters

x Single

A number representing a sine, where `x` must be greater than or equal to -1, but less than or equal to 1.

Returns

Single

An angle, θ, measured in radians, such that -π/2 ≤ θ ≤ π/2.

-or-

NaN if `x` < -1 or `x` > 1 or `x` equals NaN.

Remarks

A positive return value represents a counterclockwise angle from the x-axis; a negative return value represents a clockwise angle.

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Asinh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic sine is the specified number.

C#

```
public static float Asinh (float x);
```

Parameters

x Single

A number representing a hyperbolic sine, where `x` must be greater than or equal to NegativeInfinity, but less than or equal to PositiveInfinity.

Returns

Single

An angle, θ, measured in radians.

-or-

NaN if `x` equals NaN.

Remarks

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Atan(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose tangent is the specified number.

```
C#
public static float Atan (float x);
```

Parameters

x Single

A number representing a tangent.

Returns

Single

An angle, θ, measured in radians, such that -π/2 ≤ θ ≤ π/2.

-or-

NaN if `x` equals NaN, -π/2 rounded to double precision (-1.5707963267949) if `x` equals NegativeInfinity, or π/2 rounded to double precision (1.5707963267949) if `x` equals PositiveInfinity.

Remarks

A positive return value represents a counterclockwise angle from the x-axis; a negative return value represents a clockwise angle.

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Atan2(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose tangent is the quotient of two specified numbers.

```
C#
public static float Atan2 (float y, float x);
```

Parameters

y Single

The y coordinate of a point.

x Single

The x coordinate of a point.

Returns

Single

An angle, θ, measured in radians, such that -π ≤ θ ≤ π, and tan(θ) = `y` / `x`, where (`x`, `y`) is a point in the Cartesian plane. Observe the following:

- For (`x`, `y`) in quadrant 1, 0 < θ < π/2.
- For (`x`, `y`) in quadrant 2, π/2 < θ ≤ π.
- For (`x`, `y`) in quadrant 3, -π < θ < -π/2.
- For (`x`, `y`) in quadrant 4, -π/2 < θ < 0.

For points on the boundaries of the quadrants, the return value is the following:

- If y is 0 and x is not negative, θ = 0.
- If y is 0 and x is negative, θ = π.

- If y is positive and x is 0, θ = π/2.
- If y is negative and x is 0, θ = -π/2.
- If y is 0 and x is 0, θ = 0.

If `x` or `y` is NaN, or if `x` and `y` are either PositiveInfinity or NegativeInfinity, the method returns NaN.

Remarks

The return value is the angle in the Cartesian plane formed by the x-axis, and a vector starting from the origin, (0,0), and terminating at the point, (x,y).

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Atanh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the angle whose hyperbolic tangent is the specified number.

```
C#
public static float Atanh (float x);
```

Parameters

x Single

A number representing a hyperbolic tangent, where `x` must be greater than or equal to -1, but less than or equal to 1.

Returns

Single

An angle, θ, measured in radians, such that -∞ < θ <-1, or 1 < θ < ∞.

-or-

NaN if `x` < -1 or `x` > 1 or `x` equals NaN.

Remarks

Multiply the return value by 180/MathF.PI to convert from radians to degrees.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.BitDecrement(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the largest value that compares less than a specified value.

```
C#
public static float BitDecrement (float x);
```

Parameters

x Single

The value to decrement.

Returns

Single

The largest value that compares less than `x`.

-or-

NegativeInfinity if `x` is equals NegativeInfinity.

-or-

NaN if `x` equals NaN.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.BitIncrement(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smallest value that compares greater than a specified value.

```
C#
public static float BitIncrement (float x);
```

Parameters

x Single

The value to increment.

Returns

Single

The smallest value that compares greater than `x`.

-or-

PositiveInfinity if `x` equals PositiveInfinity.

-or-

NaN if `x` is equals NaN.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.Cbrt(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the cube root of a specified number.

```
C#
public static float Cbrt (float x);
```

Parameters

x Single

The number whose cube root is to be found.

Returns

Single

The cube root of `x`.

-or-

NaN if `x` is equals NaN.

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

See also

- Pow(Single, Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Ceiling(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smallest integral value that is greater than or equal to the specified single-precision floating-point number.

```C#
public static float Ceiling (float x);
```

Parameters

x Single

A single-precision floating-point number.

Returns

Single

The smallest integral value that is greater than or equal to `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, that value is returned. Note that this method returns a Single instead of an integral type.

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward positive infinity. In other words, if `x` is positive, the presence of any fractional component causes `x` to be rounded to the next highest integer. If `x` is negative, the rounding operation causes any fractional component of `x` to be discarded. The operation of this method differs from the Floor(Single) method, which supports rounding toward negative infinity.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Round
- Floor(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.CopySign(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value with the magnitude of `x` and the sign of `y`.

C#

```
public static float CopySign (float x, float y);
```

Parameters

x Single

A number whose magnitude is used in the result.

y Single

A number whose sign is the used in the result.

Returns

Single

A value with the magnitude of `x` and the sign of `y`.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

MathF.Cos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the cosine of the specified angle.

```
C#
public static float Cos (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The cosine of `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Cosh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic cosine of the specified angle.

C#

```
public static float Cosh (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The hyperbolic cosine of `x`. If `x` is equal to NegativeInfinity or PositiveInfinity, PositiveInfinity is returned. If `x` is equal to NaN, NaN is returned.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Exp(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns `e` raised to the specified power.

```
C#
public static float Exp (float x);
```

Parameters

x Single

A number specifying a power.

Returns

Single

The number `e` raised to the power `x`. If `x` equals NaN or PositiveInfinity, that value is returned. If `x` equals NegativeInfinity, 0 is returned.

Remarks

`e` is a mathematical constant whose value is approximately 2.71828.

Use the Pow method to calculate powers of other bases.

Exp is the inverse of Log.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- E
- Pow(Single, Single)
- Log(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Floor(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the largest integral value less than or equal to the specified single-precision floating-point number.

```
C#
public static float Floor (float x);
```

Parameters

x Single

A single-precision floating-point number.

Returns

Single

The largest integral value less than or equal to `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, that value is returned.

Remarks

The behavior of this method follows IEEE Standard 754, section 4. This kind of rounding is sometimes called rounding toward negative infinity. In other words, if `x` is positive, any fractional component is truncated. If `x` is negative, the presence of any fractional component causes it to be rounded to the smaller integer. The operation of this method differs from the Ceiling method, which supports rounding toward positive infinity.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Round
- Ceiling(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.FusedMultiplyAdd(Single, Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns (x * y) + z, rounded as one ternary operation.

```
C#
public static float FusedMultiplyAdd (float x, float y, float z);
```

Parameters

x Single

The number to be multiplied with `y`.

y Single

The number to be multiplied with `x`.

z Single

The number to be added to the result of `x` multiplied by `y`.

Returns

Single

(x * y) + z, rounded as one ternary operation.

Remarks

This computes `(x * y)` as if to infinite precision, adds `z` to that result as if to infinite precision, and finally rounds to the nearest representable value.

This differs from the non-fused sequence which would compute `(x * y)` as if to infinite preicision, round the result to the nearest representable value, add 'z' to the rounded

result as if to infinite precision, and finally round to the nearest representable value.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.IEEERemainder(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the remainder resulting from the division of a specified number by another specified number.

```
C#
public static float IEEERemainder (float x, float y);
```

Parameters

x Single

A dividend.

y Single

A divisor.

Returns

Single

A number equal to `x` - (`y` Q), where Q is the quotient of `x` / `y` rounded to the nearest integer (if `x` / `y` falls halfway between two integers, the even integer is returned).

If `x` - (`y` Q) is zero, the value +0 is returned if `x` is positive, or -0 if `x` is negative.

If `y` = 0, NaN is returned.

Remarks

This operation complies with the remainder operation defined in Section 5.1 of ANSI/IEEE Std 754-1985; IEEE Standard for Binary Floating-Point Arithmetic; Institute of

Electrical and Electronics Engineers, Inc; 1985.

The IEEERemainder method is not the same as the remainder operator. Although both return the remainder after division, the formulas they use are different. The formula for the IEEERemainder method is:

```
IEEERemainder = dividend - (divisor * MathF.Round(dividend / divisor))
```

In contrast, the formula for the remainder operator is:

```
Remainder = (MathF.Abs(dividend) - (MathF.Abs(divisor) *
            (MathF.Floor(MathF.Abs(dividend) / MathF.Abs(divisor))))) *
            MathF.Sign(dividend)
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.ILogB(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 2 integer logarithm of a specified number.

C#

```
public static int ILogB (float x);
```

Parameters

x Single

The number whose logarithm is to be found.

Returns

Int32

One of the values in the following table.

Expand table

`x` parameter	Return value
Default	The base 2 integer log of `x`; that is, (int)log2(`x`).
Zero	Int32.MinValue
Equal to NaN or PositiveInfinity or NegativeInfinity	Int32.MaxValue

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Log Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Log(Single)	Returns the natural (base `e`) logarithm of a specified number.
Log(Single, Single)	Returns the logarithm of a specified number in a specified base.

Log(Single)

Returns the natural (base `e`) logarithm of a specified number.

C#

```
public static float Log (float x);
```

Parameters

x Single

The number whose logarithm is to be found.

Returns

Single

One of the values in the following table.

Expand table

`x` parameter	Return value
Positive	The natural logarithm of `x`; that is, ln `x`, or log e `x`

`x` parameter	Return value
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

See also

- E
- Exp(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Log(Single, Single)

Returns the logarithm of a specified number in a specified base.

C#

```
public static float Log (float x, float y);
```

Parameters

x Single

The number whose logarithm is to be found.

y Single

The base.

Returns

Single

One of the values in the following table. (+Infinity denotes PositiveInfinity, -Infinity denotes NegativeInfinity, and NaN denotes NaN.)

Expand table

x	newBase	Return value
x > 0	(0 < newBase < 1) -or-(newBase > 1)	lognewBase(a)
x < 0	(any value)	NaN
(any value)	newBase < 0	NaN
x != 1	newBase = 0	NaN
x != 1	newBase = +Infinity	NaN
x = NaN	(any value)	NaN
(any value)	newBase = NaN	NaN
(any value)	newBase = 1	NaN
x = 0	0 < newBase < 1	+Infinity
x = 0	newBase > 1	-Infinity
x = +Infinity	0 < newBase < 1	-Infinity
x = +Infinity	newBase > 1	+Infinity
x = 1	newBase = 0	0
x = 1	newBase = +Infinity	0

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Log10(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 10 logarithm of a specified number.

C#

```
public static float Log10 (float x);
```

Parameters

x Single

A number whose logarithm is to be found.

Returns

Single

One of the values in the following table.

Expand table

`x` parameter	Return value
Positive	The base 10 log of `x`; that is, log 10`x`.
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Log2(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the base 2 logarithm of a specified number.

C#

```
public static float Log2 (float x);
```

Parameters

x Single

A number whose logarithm is to be found.

Returns

Single

One of the values in the following table.

Expand table

`x` parameter	Return value
Positive	The base 2 log of `x`; that is, log 2`x`.
Zero	NegativeInfinity
Negative	NaN
Equal to NaN	NaN
Equal to PositiveInfinity	PositiveInfinity

Remarks

Parameter `x` is specified as a base 10 number.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Max(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the larger of two single-precision floating-point numbers.

C#

```
public static float Max (float x, float y);
```

Parameters

x Single

The first of two single-precision floating-point numbers to compare.

y Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `x` or `y`, whichever is larger. If `x`, or `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.MaxMagnitude(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the larger magnitude of two single-precision floating-point numbers.

C#

```
public static float MaxMagnitude (float x, float y);
```

Parameters

x Single

The first of two single-precision floating-point numbers to compare.

y Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `x` or `y`, whichever has the larger magnitude. If `x`, or `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.Min(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smaller of two single-precision floating-point numbers.

C#

```
public static float Min (float x, float y);
```

Parameters

x Single

The first of two single-precision floating-point numbers to compare.

y Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `x` or `y`, whichever is smaller. If `x`, `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.MinMagnitude(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the smaller magnitude of two single-precision floating-point numbers.

C#

```
public static float MinMagnitude (float x, float y);
```

Parameters

x Single

The first of two single-precision floating-point numbers to compare.

y Single

The second of two single-precision floating-point numbers to compare.

Returns

Single

Parameter `x` or `y`, whichever has the smaller magnitude. If `x`, or `y`, or both `x` and `y` are equal to NaN, NaN is returned.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.Pow(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a specified number raised to the specified power.

C#

```
public static float Pow (float x, float y);
```

Parameters

x Single

A single-precision floating-point number to be raised to a power.

y Single

A single-precision floating-point number that specifies a power.

Returns

Single

The number `x` raised to the power `y`.

Remarks

The following table indicates the return value when various values or ranges of values are specified for the `x` and `y` parameters. For more information, see Single.PositiveInfinity, Single.NegativeInfinity, and Single.NaN.

Expand table

Parameters	Return value
`x` is any value; `y` = 0.	1
`x` = 1; `y` is any value.	1

Parameters	Return value
`x` or `y` = `NaN` (except if `y` = 0 or `x` = 1).	`NaN`
`x` = `NegativeInfinity`; `y` < 0.	0
`x` = `NegativeInfinity`; `y` is a positive odd integer.	`NegativeInfinity`
`x` = `NegativeInfinity`; `y` is positive but not an odd integer.	`PositiveInfinity`
`x` < 0 but not `NegativeInfinity`; `y` is not an integer, `NegativeInfinity`, or `PositiveInfinity`.	`NaN`
`x` = -1; `y` = `NegativeInfinity` or `PositiveInfinity`.	`NaN`
-1 < `x` < 1; `y` = `NegativeInfinity`.	`PositiveInfinity`
-1 < `x` < 1; `y` = `PositiveInfinity`.	0
`x` < -1 or `x` > 1; `y` = `NegativeInfinity`.	0
`x` < -1 or `x` > 1; `y` = `PositiveInfinity`.	`PositiveInfinity`
`x` = 0; `y` < 0.	`PositiveInfinity`
`x` = 0; `y` > 0.	0
`x` = `PositiveInfinity`; `y` < 0.	0
`x` = `PositiveInfinity`; `y` > 0.	`PositiveInfinity`

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Sqrt(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.ReciprocalEstimate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an estimate of the reciprocal of a specified number.

```
C#
public static float ReciprocalEstimate (float x);
```

Parameters

x Single

The number whose reciprocal is to be estimated.

Returns

Single

An estimate of the reciprocal of `x`.

Remarks

On x86/x64 hardware this may use the `RCPSS` instruction which has a maximum relative error of `1.5 * 2^-12`.

On ARM64 hardware this may use the `FRECPE` instruction which performs a single Newton-Raphson iteration.

On hardware without specialized support, this may just return `1.0 / x`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.ReciprocalSqrtEstimate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an estimate of the reciprocal square root of a specified number.

```
C#
public static float ReciprocalSqrtEstimate (float x);
```

Parameters

x Single

The number whose reciprocal square root is to be estimated.

Returns

Single

An estimate of the reciprocal square root `x`.

Remarks

On x86/x64 hardware this may use the `RSQRTSS` instruction which has a maximum relative error of `1.5 * 2^-12`.

On ARM64 hardware this may use the `FRSQRTE` instruction which performs a single Newton-Raphson iteration.

On hardware without specialized support, this may just return `1.0 / Sqrt(x)`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Round(Single, Int32, MidpointRounding)	Rounds a single-precision floating-point value to a specified number of fractional digits using the specified rounding convention.
Round(Single, MidpointRounding)	Rounds a single-precision floating-point value to an integer using the specified rounding convention.
Round(Single)	Rounds a single-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.
Round(Single, Int32)	Rounds a single-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.

Round(Single, Int32, MidpointRounding)

Rounds a single-precision floating-point value to a specified number of fractional digits using the specified rounding convention.

C#

```
public static float Round (float x, int digits, MidpointRounding mode);
```

Parameters

x Single

A single-precision floating-point number to be rounded.

`digits` Int32

The number of fractional digits in the return value.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Single

The number that `x` is rounded to that has `digits` fractional digits. If `x` has fewer fractional digits than `digits`, `x` is returned unchanged.

Exceptions

ArgumentOutOfRangeException

`digits` is less than 0 or greater than 6.

ArgumentException

`mode` is not a valid value of MidpointRounding.

Remarks

The value of the `digits` argument can range from 0 to 6. The maximum number of integral and fractional digits supported by the Single type is 6.

> ⓘ **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `x` argument is Single.NaN, the method returns Single.NaN. If `x` is Single.PositiveInfinity or Single.NegativeInfinity, the method returns Single.PositiveInfinity or Single.NegativeInfinity, respectively.

See also

- Ceiling(Single)

- Floor(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Round(Single, MidpointRounding)

Rounds a single-precision floating-point value to an integer using the specified rounding convention.

C#

```
public static float Round (float x, MidpointRounding mode);
```

Parameters

`x` Single

A single-precision floating-point number to be rounded.

`mode` MidpointRounding

One of the enumeration values that specifies which rounding strategy to use.

Returns

Single

The integer that `x` is rounded to using the `mode` rounding convention. This method returns a Single instead of an integral type.

Exceptions

ArgumentException

`mode` is not a valid value of MidpointRounding.

Remarks

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `x` argument is Single.NaN, the method returns Single.NaN. If `x` is Single.PositiveInfinity or Single.NegativeInfinity, the method returns Single.PositiveInfinity or Single.NegativeInfinity, respectively.

See also

- Ceiling(Single)
- Floor(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Round(Single)

Rounds a single-precision floating-point value to the nearest integral value, and rounds midpoint values to the nearest even number.

C#

```
public static float Round (float x);
```

Parameters

x Single

A single-precision floating-point number to be rounded.

Returns

Single

The integer nearest `x`. If the fractional component of `x` is halfway between two integers, one of which is even and the other odd, then the even number is returned. Note that this method returns a Single instead of an integral type.

Remarks

This method uses the default rounding convention of MidpointRounding.ToEven.

> ⓘ **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `x` argument is Single.NaN, the method returns Single.NaN. If `x` is Single.PositiveInfinity or Single.NegativeInfinity, the method returns Single.PositiveInfinity or Single.NegativeInfinity, respectively.

Notes to Callers

Because of the loss of precision that can result from representing decimal values as floating-point numbers or performing arithmetic operations on floating-point values, in some cases the Round(Single) method may not appear to round midpoint values to the nearest even integer.

See also

- Ceiling(Single)
- Floor(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Round(Single, Int32)

Rounds a single-precision floating-point value to a specified number of fractional digits, and rounds midpoint values to the nearest even number.

C#

```
public static float Round (float x, int digits);
```

Parameters

`x` Single

A single-precision floating-point number to be rounded.

`digits` Int32

The number of fractional digits in the return value.

Returns

Single

The number nearest to `x` that contains a number of fractional digits equal to `digits`.

Exceptions

ArgumentOutOfRangeException

`digits` is less than 0 or greater than 6.

Remarks

The value of the `digits` argument can range from 0 to 6. Note that 6 is the maximum number of integral and fractional digits supported by the Single type.

This method uses the default rounding convention of MidpointRounding.ToEven.

> **Important**
>
> When rounding midpoint values, the rounding algorithm performs an equality test. Because of problems of binary representation and precision in the floating-point format, the value returned by the method can be unexpected. For more information, see **Rounding and precision**.

If the value of the `x` argument is Single.NaN, the method returns Single.NaN. If `x` is Single.PositiveInfinity or Single.NegativeInfinity, the method returns Single.PositiveInfinity or Single.NegativeInfinity, respectively.

See also

- Ceiling(Single)
- Floor(Single)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.ScaleB(Single, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns x * 2^n computed efficiently.

C#

```
public static float ScaleB (float x, int n);
```

Parameters

x Single

A single-precision floating-point number that specifies the base value.

n Int32

A single-precision floating-point number that specifies the power.

Returns

Single

x * 2^n computed efficiently.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

MathF.Sign(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an integer that indicates the sign of a single-precision floating-point number.

C#

```
public static int Sign (float x);
```

Parameters

x Single

A signed number.

Returns

Int32

A number that indicates the sign of `x`, as shown in the following table.

Expand table

Return value	Meaning
-1	`x` is less than zero.
0	`x` is equal to zero.
1	`x` is greater than zero.

Exceptions

ArithmeticException

`x` is equal to NaN.

Remarks

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Sin(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the sine of the specified angle.

C#

```
public static float Sin (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The sine of `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.SinCos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the sine and cosine of the specified angle.

```
C#
public static (float Sin, float Cos) SinCos (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

ValueTuple<Single,Single>

The sine and cosine of `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Remarks

The angle, `x`, must be in radians. Multiply by Math.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MathF.Sinh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic sine of the specified angle.

C#

```
public static float Sinh (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The hyperbolic sine of `x`. If `x` is equal to NegativeInfinity, PositiveInfinity, or NaN, this method returns a Single equal to `x`.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Sqrt(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the square root of a specified number.

C#

```
public static float Sqrt (float x);
```

Parameters

x Single

The number whose square root is to be found.

Returns

Single

One of the values in the following table.

Expand table

`x` parameter	Return value
Zero or positive	The positive square root of `x`.
Negative	NaN
Equals NaN	NaN
Equals PositiveInfinity	PositiveInfinity

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

See also

- Pow(Single, Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Tan(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the tangent of the specified angle.

```
C#
public static float Tan (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The tangent of `x`. If `x` is equal to NaN, NegativeInfinity, or PositiveInfinity, this method returns NaN.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Tanh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hyperbolic tangent of the specified angle.

```
C#
public static float Tanh (float x);
```

Parameters

x Single

An angle, measured in radians.

Returns

Single

The hyperbolic tangent of `x`. If `x` is equal to NegativeInfinity, this method returns -1. If value is equal to PositiveInfinity, this method returns 1. If `x` is equal to NaN, this method returns NaN.

Remarks

The angle, `x`, must be in radians. Multiply by MathF.PI/180 to convert degrees to radians.

This method calls into the underlying C runtime, and the exact result or valid input range may differ between different operating systems or architectures.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MathF.Truncate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the integral part of a specified single-precision floating-point number.

C#

```
public static float Truncate (float x);
```

Parameters

x Single

A number to truncate.

Returns

Single

The integral part of `x`; that is, the number that remains after any fractional digits have been discarded, or one of the values listed in the following table.

Expand table

x	Return value
NaN	NaN
NegativeInfinity	NegativeInfinity
PositiveInfinity	PositiveInfinity

Remarks

Truncate rounds `x` to the nearest integer towards zero.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Round
- Ceiling(Single)
- Floor(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemberAccessException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an attempt to access a class member fails.

```
C#
public class MemberAccessException : SystemException
```

Inheritance Object → Exception → SystemException → MemberAccessException

Derived System.FieldAccessException
System.MethodAccessException
System.MissingMemberException

Remarks

MemberAccessException is the base class for FieldAccessException, MethodAccessException, MissingMemberException, MissingMethodException, and MissingFieldException, and (.NET Native only). These exceptions are thrown when a class member is not found or access to the member is not permitted.

MemberAccessException is thrown by the following methods:

Invoke of the System.Reflection classes, GetValue of the FieldInfo class, and DynamicInvoke of the Delegate class.

Apps compiled by using the .NET Native tool chain may throw a MissingRuntimeArtifactException exception at run time. `MissingRuntimeArtifactException` is an internal-only exception type derived from MemberAccessException. The exception indicates that metadata needed for the successful execution of an app is not present at run time. You should not use a `try`/`catch` block to handle the exception. Instead, you should determine what metadata is missing and modify your app's runtime directives file to ensure that it is present at run time.

MemberAccessException uses the HRESULT COR_E_MEMBERACCESS, that has the value 0x8013151A.

For a list of initial property values for an instance of MemberAccessException, see the MemberAccessException constructors.

Constructors

Expand table

MemberAccessException()	Initializes a new instance of the MemberAccessException class.
MemberAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MemberAccessException class with serialized data.
MemberAccessException(String)	Initializes a new instance of the MemberAccessException class with a specified error message.
MemberAccessException(String, Exception)	Initializes a new instance of the MemberAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error.

	(Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception.

(Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemberAccessException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MemberAccessException class.

Overloads

Expand table

MemberAccessException()	Initializes a new instance of the MemberAccessException class.
MemberAccessException(String)	Initializes a new instance of the MemberAccessException class with a specified error message.
MemberAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MemberAccessException class with serialized data.
MemberAccessException(String, Exception)	Initializes a new instance of the MemberAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

MemberAccessException()

Initializes a new instance of the MemberAccessException class.

C#

```
public MemberAccessException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Cannot access member." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MemberAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MemberAccessException(String)

Initializes a new instance of the MemberAccessException class with a specified error message.

C#

```
public MemberAccessException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of `message` should be understood by the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MemberAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MemberAccessException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the MemberAccessException class with serialized data.

```C#
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected MemberAccessException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MemberAccessException(String, Exception)

Initializes a new instance of the MemberAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public MemberAccessException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MemberAccessException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a contiguous region of memory.

```
C#
public readonly struct Memory<T> : IEquatable<Memory<T>>
```

Type Parameters

T

The type of items in the Memory<T>.

Inheritance Object → ValueType → Memory<T>

Implements IEquatable<Memory<T>>

Remarks

Like Span<T>, `Memory<T>` represents a contiguous region of memory. Unlike Span<T>, however, `Memory<T>` is not a ref struct. This means that `Memory<T>` can be placed on the managed heap, whereas Span<T> cannot. As a result, the `Memory<T>` structure does not have the same restrictions as a Span<T> instance. In particular:

- It can be used as a field in a class.
- It can be used across `await` and `yield` boundaries.

In addition to `Memory<T>`, you can use System.ReadOnlyMemory<T> to represent immutable or read-only memory.

Constructors

Expand table

Memory<T>(T[])	Creates a new Memory<T> object over the entirety of a specified array.
Memory<T>(T[], Int32, Int32)	Creates a new Memory<T> object that includes a specified number of elements of an array beginning at a specified index.

Properties

Expand table

Empty	Returns an empty Memory<T> object.
IsEmpty	Indicates whether the current instance is empty.
Length	Gets the number of items in the current instance.
Span	Returns a span from the current instance.

Methods

Expand table

CopyTo(Memory<T>)	Copies the contents of a Memory<T> object into a destination Memory<T> object.
Equals(Memory<T>)	Determines whether the specified Memory<T> object is equal to the current object.
Equals(Object)	Determines whether the specified object is equal to the current object.
GetHashCode()	Returns the hash code for this instance.
Pin()	Creates a handle for the Memory<T> object.
Slice(Int32)	Forms a slice out of the current memory that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current memory starting at a specified index for a specified length.
ToArray()	Copies the contents from the memory into a new array.
ToString()	Returns the string representation of this Memory<T> object.
TryCopy To(Memory<T>)	Copies the contents of the memory into a destination Memory<T> instance.

Operators

Expand table

Implicit(ArraySegment<T> to Memory<T>)	Defines an implicit conversion of an ArraySegment<T> object to a Memory<T> object.
Implicit(Memory<T> to ReadOnlyMemory<T>)	Defines an implicit conversion of a Memory<T> object to a ReadOnlyMemory<T> object.
Implicit(T[] to Memory<T>)	Defines an implicit conversion of an array to a Memory<T> object.

Extension Methods

Expand table

Trim<T>(Memory<T>, T)	Removes all leading and trailing occurrences of a specified element from a memory region.
Trim<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a memory region.
TrimEnd<T>(Memory<T>, T)	Removes all trailing occurrences of a specified element from a character memory region.
TrimEnd<T>(Memory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a memory region.
TrimStart<T>(Memory<T>, T)	Removes all leading occurrences of a specified element from a memory region.
TrimStart<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Memory- and span-related types
- Memory<T> and Span<T> usage guidelines

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Memory<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Memory<T>(T[])	Creates a new Memory<T> object over the entirety of a specified array.
Memory<T>(T[], Int32, Int32)	Creates a new Memory<T> object that includes a specified number of elements of an array beginning at a specified index.

Memory<T>(T[])

Creates a new Memory<T> object over the entirety of a specified array.

C#

```
public Memory (T[]? array);
```

Parameters

array T[]

The array from which to create the Memory<T> object.

Exceptions

ArrayTypeMismatchException

`T` is a reference type, and `array` is not an array of type `T`.

-or-

The array is covariant.

Remarks

If `array` is `null`, this constructor returns a Memory<T> object with a `default<T>` value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Memory<T>(T[], Int32, Int32)

Creates a new Memory<T> object that includes a specified number of elements of an array beginning at a specified index.

```
C#

public Memory (T[]? array, int start, int length);
```

Parameters

`array` T[]

The source array.

`start` Int32

The index of the first element to include in the new Memory<T>.

`length` Int32

The number of elements to include in the new Memory<T>.

Exceptions

ArgumentOutOfRangeException

`array` is `null`, but `start` or `length` is non-zero.

-or-

`start` is outside the bounds of the array.

-or-

`start` and `length` exceeds the number of elements in the array.

ArrayTypeMismatchException

`T` is a reference type, and `array` is not an array of type `T`.

Remarks

If `array` is `null`, this constructor returns a Memory<T> object with a `default<T>` value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.Empty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an empty Memory<T> object.

C#

```
public static Memory<T> Empty { get; }
```

Property Value

Memory<T>

An empty object.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.IsEmpty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current instance is empty.

C#

```
public bool IsEmpty { get; }
```

Property Value

Boolean

`true` if the current instance is empty; otherwise, `false`.

Remarks

A Memory<T> object is empty if its length is 0.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Memory<T>.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of items in the current instance.

C#

```
public int Length { get; }
```

Property Value

Int32

The number of items in the current instance.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.Span Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a span from the current instance.

```
C#
public Span<T> Span { get; }
```

Property Value

Span<T>

A span created from the current Memory<T> object.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.CopyTo(Memory<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of a Memory<T> object into a destination Memory<T> object.

```
C#
public void CopyTo (Memory<T> destination);
```

Parameters

destination Memory<T>

The destination Memory<T> object.

Exceptions

ArgumentException

The length of `destination` is less than the length of the current instance.

Remarks

This method copies all of the contents of the current Memory<T> instance to `destination` even if the contents of the current instance and `destination` overlap.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Memory<T>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Memory<T>)	Determines whether the specified Memory<T> object is equal to the current object.
Equals(Object)	Determines whether the specified object is equal to the current object.

Equals(Memory<T>)

Determines whether the specified Memory<T> object is equal to the current object.

C#

```
public bool Equals (Memory<T> other);
```

Parameters

`other` Memory<T>

The object to compare with the current instance.

Returns

Boolean

`true` if the current instance and `other` are equal; otherwise, `false`.

Implements

Equals(T)

Remarks

The two objects are equal if:

- `other` is a Memory<T> or ReadOnlyMemory<T> object.
- Both objects point to the same array and have the same length.

The `Equals(Memory<T>)` method performs a test for reference equality; it does not compare the elements of Memory<T> for equality.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Equals(Object)

Determines whether the specified object is equal to the current object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with the current instance.

Returns

Boolean

`true` if the current instance and `obj` are equal; otherwise, `false`.

Remarks

Two Memory<T> objects are equal if both objects point to the same array and have the same length. Note that the test for equality does not check whether the contents are equal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.Pin Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a handle for the Memory<T> object.

```
C#
public System.Buffers.MemoryHandle Pin ();
```

Returns

MemoryHandle

A handle for the Memory<T> object.

Exceptions

ArgumentException

An instance with non-primitive (non-blittable) members cannot be pinned.

Remarks

The garbage collector will not move the memory until the returned MemoryHandle object is disposed. This enables you to retrieve and use the memory's address.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- IsPrimitive

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.Slice Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Slice(Int32)	Forms a slice out of the current memory that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current memory starting at a specified index for a specified length.

Slice(Int32)

Forms a slice out of the current memory that begins at a specified index.

C#

```
public Memory<T> Slice (int start);
```

Parameters

`start` Int32

The index at which to begin the slice.

Returns

Memory<T>

An object that contains all elements of the current instance from `start` to the end of the instance.

Exceptions

ArgumentOutOfRangeException

`start` is less than zero or greater than Length.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Slice(Int32, Int32)

Forms a slice out of the current memory starting at a specified index for a specified length.

```
C#
public Memory<T> Slice (int start, int length);
```

Parameters

`start` Int32

The index at which to begin the slice.

`length` Int32

The number of elements to include in the slice.

Returns

Memory<T>

An object that contains `length` elements from the current instance starting at `start`.

Exceptions

ArgumentOutOfRangeException

`start` is less than zero or greater than Length.

-or-

`length` is greater than Length - `start`

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Memory<T>.ToArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents from the memory into a new array.

C#

```
public T[] ToArray ();
```

Returns

T[]

An array containing the elements in the current memory.

Remarks

Because a call to the `ToArray` method performs a heap allocation, it should generally be avoided. However, it is sometimes necessary to take advantage of functionality that is only available for arrays.

Each call to the `ToArray` method returns a new array.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Memory<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of this Memory<T> object.

C#

```
public override string ToString ();
```

Returns

String

the string representation of this Memory<T> object.

Remarks

For a `Memory<Char>`, the `ToString` method returns a String that contains the characters pointed to by the Memory<T>. Otherwise, it returns a String with the name of the type and the number of elements that the Memory<T> contains.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Memory<T>.TryCopyTo(Memory<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of the memory into a destination Memory<T> instance.

C#

```
public bool TryCopyTo (Memory<T> destination);
```

Parameters

destination Memory<T>

The destination Memory<T> object.

Returns

Boolean

`true` if the copy operation succeeds; otherwise, `false`.

Remarks

This method copies all of the contents of the current Memory<T> instance to `destination` even if the contents of the current instance and `destination` overlap.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Memory<T>.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(T[] to Memory<T>)	Defines an implicit conversion of an array to a Memory<T> object.
Implicit(ArraySegment<T> to Memory<T>)	Defines an implicit conversion of an ArraySegment<T> object to a Memory<T> object.
Implicit(Memory<T> to ReadOnlyMemory<T>)	Defines an implicit conversion of a Memory<T> object to a ReadOnlyMemory<T> object.

Implicit(T[] to Memory<T>)

Defines an implicit conversion of an array to a Memory<T> object.

C#

```
public static implicit operator Memory<T> (T[]? array);
```

Parameters

`array` T[]

The array to convert.

Returns

Memory<T>

The converted object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(ArraySegment<T> to Memory<T>)

Defines an implicit conversion of an ArraySegment<T> object to a Memory<T> object.

```
C#

public static implicit operator Memory<T> (ArraySegment<T> segment);
```

Parameters

`segment` ArraySegment<T>

The object to convert.

Returns

Memory<T>

The converted ArraySegment<T> object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(Memory<T> to ReadOnlyMemory<T>)

Defines an implicit conversion of a Memory<T> object to a ReadOnlyMemory<T> object.

```
C#

public static implicit operator ReadOnlyMemory<T> (Memory<T> memory);
```

Parameters

`memory` Memory<T>

The object to convert.

Returns

ReadOnlyMemory<T>

The converted object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions Class

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Provides extension methods for the memory-related and span-related types, such as Memory<T>, ReadOnlyMemory<T>, Span<T>, and ReadOnlySpan<T>.

C#

```
public static class MemoryExtensions
```

Inheritance Object → MemoryExtensions

Methods

Expand table

AsMemory(String)	Creates a new `ReadOnlyMemory<Char>` over the portion of the target string.
AsMemory(String, Index)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified index.
AsMemory(String, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified character position.
AsMemory(String, Int32, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string beginning at a specified position with a length.
AsMemory(String, Range)	Creates a new `ReadOnlyMemory<Char>` over a specified range of the target string.
AsMemory<T>(ArraySegment<T>)	Creates a new memory region over the portion of the target array segment.

AsMemory<T>(ArraySegment<T>, Int32)	Creates a new memory region over the portion of the target array segment starting at a specified position to the end of the segment.
AsMemory<T>(ArraySegment<T>, Int32, Int32)	Creates a new memory region over the portion of the target array segment beginning at a specified position with a specified length.
AsMemory<T>(T[])	Creates a new memory region over the target array.
AsMemory<T>(T[], Index)	Creates a new memory region over the portion of the target array starting at a specified index to the end of the array.
AsMemory<T>(T[], Int32)	Creates a new memory region over the portion of the target array starting at a specified position to the end of the array.
AsMemory<T>(T[], Int32, Int32)	Creates a new memory region over the portion of the target array beginning at a specified position with a specified length.
AsMemory<T>(T[], Range)	Creates a new memory region over the portion of the target array beginning at inclusive start index of the range and ending at the exclusive end index of the range.
AsSpan(String)	Creates a new read-only span over a string.
AsSpan(String, Index)	Creates a new ReadOnlySpan<T> over a portion of the target string from a specified position to the end of the string.
AsSpan(String, Int32)	Creates a new read-only span over a portion of the target string from a specified position to the end of the string.
AsSpan(String, Int32, Int32)	Creates a new read-only span over a portion of the target string from a specified position for a specified number of characters.
AsSpan(String, Range)	Creates a new ReadOnlySpan<T> over a portion of a target string using the range start and end indexes.
AsSpan<T>(ArraySegment<T>)	Creates a new span over a target array segment.
AsSpan<T>(ArraySegment<T>, Index)	Creates a new span over a portion of the target array segment beginning at a specified

	index and ending at the end of the segment.
AsSpan<T>(ArraySegment<T>, Int32)	Creates a new span over a portion of a target array segment from a specified position to the end of the segment.
AsSpan<T>(ArraySegment<T>, Int32, Int32)	Creates a new span over a portion of a target array segment from a specified position for a specified length.
AsSpan<T>(ArraySegment<T>, Range)	Creates a new span over a portion of a target array segment using the range start and end indexes.
AsSpan<T>(T[])	Creates a new span over a target array.
AsSpan<T>(T[], Index)	Creates a new span over the portion of the target array defined by an Index value.
AsSpan<T>(T[], Int32)	Creates a new span over a portion of the target array starting at a specified position to the end of the array.
AsSpan<T>(T[], Int32, Int32)	Creates a new span over the portion of the target array beginning at a specified position for a specified length.
AsSpan<T>(T[], Range)	Creates a new span over a portion of a target array defined by a Range value.
BinarySearch<T,TComparable>(ReadOnlySpan<T>, TComparable)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified `TComparable` generic type.
BinarySearch<T,TComparable>(Span<T>, TComparable)	Searches an entire sorted Span<T> for a value using the specified `TComparable` generic type.
BinarySearch<T,TComparer>(ReadOnlySpan<T>, T, TComparer)	Searches an entire sorted ReadOnlySpan<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T,TComparer>(Span<T>, T, TComparer)	Searches an entire sorted Span<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T>(ReadOnlySpan<T>, IComparable<T>)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified IComparable<T> generic interface.
BinarySearch<T>(Span<T>, IComparable<T>)	Searches an entire sorted Span<T> for a value using the specified IComparable<T> generic interface.

CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Finds the length of any common prefix shared between `span` and `other`.
CompareTo(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Compares one character span with another using a specified string comparison, and returns an integer that indicates their relative position in the sort order.
Contains(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Indicates whether a specified value occurs within a read-only character span.
Contains<T>(ReadOnlySpan<T>, T)	Indicates whether a specified value is found in a read-only span. Values are compared using IEquatable{T}.Equals(T).
Contains<T>(Span<T>, T)	Indicates whether a specified value is found in a span. Values are compared using IEquatable{T}.Equals(T).
ContainsAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If

	not found, returns `false`.
ContainsAny<T>(Span<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for any value other than the specified `value`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(Span<T>, T)	Searches for any value other than the specified `value`.
ContainsAnyExcept<T>(Span<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(Span<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
ContainsAnyExceptInRange<T>(Span<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
ContainsAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive,

	and returns `true` if found. If not found, returns `false`.
ContainsAnyInRange<T>(Span<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.
CopyTo<T>(T[], Memory<T>)	Copies the contents of the array into a memory region.
CopyTo<T>(T[], Span<T>)	Copies the contents of the array into the span.
Count<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Counts the number of times `value` occurs in `span`.
Count<T>(ReadOnlySpan<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(Span<T>, ReadOnlySpan<T>)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(Span<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
EndsWith(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Determines whether the end of the `span` matches the specified `value` when compared using the specified `comparisonType` option.
EndsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a read-only span.
EndsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a span.
EnumerateLines(ReadOnlySpan<Char>)	Returns an enumeration of lines over the provided span.
EnumerateLines(Span<Char>)	Returns an enumeration of lines over the provided span.
EnumerateRunes(ReadOnlySpan<Char>)	Returns an enumeration of Rune from the provided read-only span.
EnumerateRunes(Span<Char>)	Returns an enumeration of Rune from the provided span.
Equals(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Determines whether this `span` and the specified `other` span have the same characters when compared using the specified `comparisonType` option.

IndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Reports the zero-based index of the first occurrence of the specified `value` in the current `span`.
IndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(Span<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

IndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.
IndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(Span<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.
IndexOfAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyExceptInRange<T>(Span<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
IsWhiteSpace(ReadOnlySpan<Char>)	Indicates whether the specified span contains only whitespace characters.
LastIndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Reports the zero-based index of the last occurrence of the specified `value` in the

	current `span`.
LastIndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(Span<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.
LastIndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(Span<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.
LastIndexOfAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyExceptInRange<T>(Span<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences overlap in memory.
Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, Int32)	Determines whether two read-only sequences overlap in memory and outputs the element offset.

Overlaps<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span overlap in memory.
Overlaps<T>(Span<T>, ReadOnlySpan<T>, Int32)	Determines whether a span and a read-only span overlap in memory and outputs the element offset.
Replace<T>(ReadOnlySpan<T>, Span<T>, T, T)	Copies `source` to `destination`, replacing all occurrences of `oldValue` with `newValue`.
Replace<T>(Span<T>, T, T)	Replaces all occurrences of `oldValue` with `newValue`.
Reverse<T>(Span<T>)	Reverses the sequence of the elements in the entire span.
SequenceCompareTo<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines the relative order of two read-only sequences by comparing their elements using IComparable{T}.CompareTo(T).
SequenceCompareTo<T>(Span<T>, ReadOnlySpan<T>)	Determines the relative order of a span and a read-only span by comparing the elements using IComparable{T}.CompareTo(T).
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences are equal by comparing the elements using IEquatable{T}.Equals(T).
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
SequenceEqual<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span are equal by comparing the elements using IEquatable{T}.Equals(T).
SequenceEqual<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
Sort<T,TComparer>(Span<T>, TComparer)	Sorts the elements in the entire Span<T> using the `TComparer`.
Sort<T>(Span<T>)	Sorts the elements in the entire Span<T> using the IComparable<T> implementation of each element of the Span<T>.
Sort<T>(Span<T>, Comparison<T>)	Sorts the elements in the entire Span<T> using the specified Comparison<T>.
Sort<TKey,TValue,TComparer>(Span<TKey>, Span<TValue>, TComparer)	Sorts a pair of spans (one containing the keys and the other containing the corresponding

	items) based on the keys in the first Span<T> using the specified comparer.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the IComparable<T> implementation of each key.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>, Comparison<TKey>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparison.
Split(ReadOnlySpan<Char>, Span<Range>, Char, StringSplitOptions)	Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.
Split(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.
SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.
SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<String>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.
StartsWith(ReadOnlySpan<Char>, ReadOnly Span<Char>, StringComparison)	Determines whether a read-only character span begins with a specified value when compared using a specified StringComparison value.
StartsWith<T>(ReadOnlySpan<T>, ReadOnly Span<T>)	Determines whether a specified sequence appears at the start of a read-only span.
StartsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a specified sequence appears at the start of a span.
ToLower(ReadOnlySpan<Char>, Span<Char>, CultureInfo)	Copies the characters from the source span into the destination, converting each character

	to lowercase, using the casing rules of the specified culture.
ToLowerInvariant(ReadOnlySpan<Char>, Span<Char>)	Copies the characters from the source span into the destination, converting each character to lowercase, using the casing rules of the invariant culture.
ToUpper(ReadOnlySpan<Char>, Span<Char>, CultureInfo)	Copies the characters from the source span into the destination, converting each character to uppercase, using the casing rules of the specified culture.
ToUpperInvariant(ReadOnlySpan<Char>, Span<Char>)	Copies the characters from the source span into the destination, converting each character to uppercase using the casing rules of the invariant culture.
Trim(Memory<Char>)	Removes all leading and trailing whitespace characters from a character memory region.
Trim(ReadOnlyMemory<Char>)	Removes all leading and trailing whitespace characters from a read-only character memory region.
Trim(ReadOnlySpan<Char>)	Removes all leading and trailing whitespace characters from a read-only character span.
Trim(ReadOnlySpan<Char>, Char)	Removes all leading and trailing occurrences of a specified character from a read-only character span.
Trim(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Removes all leading and trailing occurrences of a set of characters specified in a read-only span from a read-only character span.
Trim(Span<Char>)	Removes all leading and trailing whitespace characters from a character span.
Trim<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a memory region.
Trim<T>(Memory<T>, T)	Removes all leading and trailing occurrences of a specified element from a memory region.
Trim<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
Trim<T>(ReadOnlyMemory<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only

	memory region.
Trim<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only span.
Trim<T>(ReadOnlySpan<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only span.
Trim<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a span.
Trim<T>(Span<T>, T)	Removes all leading and trailing occurrences of a specified element from a span.
TrimEnd(Memory<Char>)	Removes all trailing whitespace characters from a character memory region.
TrimEnd(ReadOnlyMemory<Char>)	Removes all trailing whitespace characters from a read-only character memory region.
TrimEnd(ReadOnlySpan<Char>)	Removes all trailing whitespace characters from a read-only character span.
TrimEnd(ReadOnlySpan<Char>, Char)	Removes all trailing occurrences of a specified character from a read-only span.
TrimEnd(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Removes all trailing occurrences of a set of characters specified in a read-only span from a read-only character span.
TrimEnd(Span<Char>)	Removes all trailing whitespace characters from a character span.
TrimEnd<T>(Memory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a memory region.
TrimEnd<T>(Memory<T>, T)	Removes all trailing occurrences of a specified element from a character memory region.
TrimEnd<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
TrimEnd<T>(ReadOnlyMemory<T>, T)	Removes all trailing occurrences of a specified element from a read-only memory region.
TrimEnd<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only span.

TrimEnd<T>(ReadOnlySpan<T>, T)	Removes all trailing occurrences of a specified element from a read-only span.
TrimEnd<T>(Span<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a span.
TrimEnd<T>(Span<T>, T)	Removes all trailing occurrences of a specified element from a span.
TrimStart(Memory<Char>)	Removes all leading whitespace characters from a memory region.
TrimStart(ReadOnlyMemory<Char>)	Removes all leading whitespace characters from a read-only memory region.
TrimStart(ReadOnlySpan<Char>)	Removes all leading whitespace characters from a read-only span.
TrimStart(ReadOnlySpan<Char>, Char)	Removes all leading occurrences of a specified character from the span.
TrimStart(ReadOnlySpan<Char>, ReadOnly Span<Char>)	Removes all leading occurrences of a set of characters specified in a read-only span from the span.
TrimStart(Span<Char>)	Removes all leading whitespace characters from a span.
TrimStart<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.
TrimStart<T>(Memory<T>, T)	Removes all leading occurrences of a specified element from a memory region.
TrimStart<T>(ReadOnlyMemory<T>, ReadOnly Span<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.
TrimStart<T>(ReadOnlyMemory<T>, T)	Removes all leading occurrences of a specified element from a memory region.
TrimStart<T>(ReadOnlySpan<T>, ReadOnly Span<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from the span.
TrimStart<T>(ReadOnlySpan<T>, T)	Removes all leading occurrences of a specified element from the span.
TrimStart<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from

	the span.
TrimStart<T>(Span<T>, T)	Removes all leading occurrences of a specified element from the span.
TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, Object[])	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, ReadOnlySpan<Object>)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite(Span<Char>, IFormatProvider, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)	Writes the specified interpolated string to the character span.
TryWrite(Span<Char>, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)	Writes the specified interpolated string to the character span.
TryWrite<TArg0,TArg1,TArg2>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1, TArg2)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite<TArg0,TArg1>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite<TArg0>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.AsMemory Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

AsMemory(String)	Creates a new `ReadOnlyMemory<Char>` over the portion of the target string.
AsMemory(String, Index)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified index.
AsMemory(String, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified character position.
AsMemory(String, Range)	Creates a new `ReadOnlyMemory<Char>` over a specified range of the target string.
AsMemory(String, Int32, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string beginning at a specified position with a length.
AsMemory<T>(ArraySegment<T>)	Creates a new memory region over the portion of the target array segment.
AsMemory<T>(T[])	Creates a new memory region over the target array.
AsMemory<T>(ArraySegment<T>, Int32)	Creates a new memory region over the portion of the target array segment starting at a specified position to the end of the segment.
AsMemory<T>(T[], Index)	Creates a new memory region over the portion of the target array starting at a specified index to the end of the array.
AsMemory<T>(T[], Int32)	Creates a new memory region over the portion of the target array starting at a specified position to the end of the array.
AsMemory<T>(T[], Range)	Creates a new memory region over the portion of the target array beginning at inclusive start index of the range and ending at the exclusive end index of the range.

AsMemory<T> (ArraySegment<T>, Int32, Int32)	Creates a new memory region over the portion of the target array segment beginning at a specified position with a specified length.
AsMemory<T>(T[], Int32, Int32)	Creates a new memory region over the portion of the target array beginning at a specified position with a specified length.

AsMemory(String)

Creates a new `ReadOnlyMemory<Char>` over the portion of the target string.

```
C#
public static ReadOnlyMemory<char> AsMemory (this string? text);
```

Parameters

`text` String

The target string.

Returns

ReadOnlyMemory<Char>

The read-only character memory representation of the string, or `default` if `text` is `null`.

Remarks

Returns default when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory(String, Index)

Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified index.

C#

```
public static ReadOnlyMemory<char> AsMemory (this string? text, Index
startIndex);
```

Parameters

`text` String

The target string.

`startIndex` Index

The index at which to begin this slice.

Returns

ReadOnlyMemory<Char>

The read-only character memory representation of the string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory(String, Int32)

Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified character position.

C#

```
public static ReadOnlyMemory<char> AsMemory (this string? text, int
start);
```

Parameters

`text` String

The target string.

`start` Int32

The index at which to begin this slice.

Returns

ReadOnlyMemory<Char>

The read-only character memory representation of the string, or `default` if `text` is `null`.

Exceptions

ArgumentOutOfRangeException

`start` is not in the range of `text` (`start` is < 0 or > `text.Length`).

Remarks

Returns default when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory(String, Range)

Creates a new `ReadOnlyMemory<Char>` over a specified range of the target string.

```
C#

public static ReadOnlyMemory<char> AsMemory (this string? text, Range
range);
```

Parameters

`text` String

The target string.

`range` Range

The range that indicates the start and length of the sliced string.

Returns

ReadOnlyMemory<Char>

The read-only character memory representation of the string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory(String, Int32, Int32)

Creates a new `ReadOnlyMemory<Char>` over a portion of the target string beginning at a specified position with a length.

```
C#

public static ReadOnlyMemory<char> AsMemory (this string? text, int
start, int length);
```

Parameters

`text` String

The target string.

`start` Int32

The index at which to begin this slice.

`length` Int32

The desired length for the slice.

Returns

ReadOnlyMemory<Char>

The read-only character memory representation of the string, or `default` if `text` is `null`.

Exceptions

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `text`.

Remarks

Returns default when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(ArraySegment<T>)

Creates a new memory region over the portion of the target array segment.

C#

```
public static Memory<T> AsMemory<T> (this ArraySegment<T> segment);
```

Type Parameters

`T`

The type of the segment.

Parameters

`segment` ArraySegment<T>

The segment to convert.

Returns

Memory<T>

The memory representation of the segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(T[])

Creates a new memory region over the target array.

C#

```
public static Memory<T> AsMemory<T> (this T[]? array);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The array to convert.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(ArraySegment<T>, Int32)

Creates a new memory region over the portion of the target array segment starting at a specified position to the end of the segment.

C#

```
public static Memory<T> AsMemory<T> (this ArraySegment<T> segment, int
start);
```

Type Parameters

`T`

The type of the array.

Parameters

`segment` ArraySegment<T>

The target array segment.

`start` Int32

The index at which to begin the memory.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Exceptions

ArrayTypeMismatchException

`segment` is covariant, and the type of `segment` is not exactly `T[]`.

ArgumentOutOfRangeException

`start` is less than 0 or greater than `segment.Count`.

Remarks

Returns `default` when `segment` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(T[], Index)

Creates a new memory region over the portion of the target array starting at a specified index to the end of the array.

C#

```
public static Memory<T> AsMemory<T> (this T[]? array, Index startIndex);
```

Type Parameters

T

The type of the array.

Parameters

`array` T[]

The array to convert.

`startIndex` Index

The first position of the array.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(T[], Int32)

Creates a new memory region over the portion of the target array starting at a specified position to the end of the array.

C#

```
public static Memory<T> AsMemory<T> (this T[]? array, int start);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The target array.

`start` Int32

The index at which to begin the memory.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Exceptions

ArrayTypeMismatchException

`array` is covariant, and the array's type is not exactly `T[]`.

ArgumentOutOfRangeException

`start` index less than 0 or greater than `array.Length`.

Remarks

Returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(T[], Range)

Creates a new memory region over the portion of the target array beginning at inclusive start index of the range and ending at the exclusive end index of the range.

```
C#
public static Memory<T> AsMemory<T> (this T[]? array, Range range);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The array to convert.

`range` Range

The range to convert from the array.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(ArraySegment<T>, Int32, Int32)

Creates a new memory region over the portion of the target array segment beginning at a specified position with a specified length.

C#

```
public static Memory<T> AsMemory<T> (this ArraySegment<T> segment, int
start, int length);
```

Type Parameters

`T`

The type of the array.

Parameters

`segment` ArraySegment<T>

The target array segment.

`start` Int32

The index at which to begin the memory.

`length` Int32

The number of items in the memory.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Exceptions

ArrayTypeMismatchException

`segment` is covariant, and the array's type is not exactly `T[]`.

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `segment`.

Remarks

Returns `default` when `segment` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsMemory<T>(T[], Int32, Int32)

Creates a new memory region over the portion of the target array beginning at a specified position with a specified length.

```
C#

public static Memory<T> AsMemory<T> (this T[]? array, int start, int
length);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The target array.

`start` Int32

The index at which to begin the memory region.

`length` Int32

The number of items in the memory region.

Returns

Memory<T>

The memory representation of the whole or part of the array.

Exceptions

ArrayTypeMismatchException

`array` is covariant, and the array's type is not exactly `T[]`.

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `array`.

Remarks

Returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.AsSpan Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

AsSpan(String, Int32, Int32)	Creates a new read-only span over a portion of the target string from a specified position for a specified number of characters.
AsSpan(String, Range)	Creates a new ReadOnlySpan<T> over a portion of a target string using the range start and end indexes.
AsSpan(String, Int32)	Creates a new read-only span over a portion of the target string from a specified position to the end of the string.
AsSpan(String, Index)	Creates a new ReadOnlySpan<T> over a portion of the target string from a specified position to the end of the string.
AsSpan(String)	Creates a new read-only span over a string.
AsSpan<T> (ArraySegment<T>)	Creates a new span over a target array segment.
AsSpan<T>(T[])	Creates a new span over a target array.
AsSpan<T> (ArraySegment<T>, Index)	Creates a new span over a portion of the target array segment beginning at a specified index and ending at the end of the segment.
AsSpan<T> (ArraySegment<T>, Int32)	Creates a new span over a portion of a target array segment from a specified position to the end of the segment.
AsSpan<T> (ArraySegment<T>, Range)	Creates a new span over a portion of a target array segment using the range start and end indexes.
AsSpan<T>(T[], Index)	Creates a new span over the portion of the target array defined by an Index value.
AsSpan<T>(T[], Int32)	Creates a new span over a portion of the target array starting at a specified position to the end of the array.

AsSpan<T>(T[], Range)	Creates a new span over a portion of a target array defined by a Range value.
AsSpan<T>(ArraySegment<T>, Int32, Int32)	Creates a new span over a portion of a target array segment from a specified position for a specified length.
AsSpan<T>(T[], Int32, Int32)	Creates a new span over the portion of the target array beginning at a specified position for a specified length.

AsSpan(String, Int32, Int32)

Creates a new read-only span over a portion of the target string from a specified position for a specified number of characters.

C#

```
public static ReadOnlySpan<char> AsSpan (this string? text, int start,
int length);
```

Parameters

`text` String

The target string.

`start` Int32

The index at which to begin this slice.

`length` Int32

The desired length for the slice.

Returns

ReadOnlySpan<Char>

The read-only span representation of the string.

Exceptions

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `text`.

Remarks

Returns `default` when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan(String, Range)

Creates a new ReadOnlySpan<T> over a portion of a target string using the range start and end indexes.

C#

```
public static ReadOnlySpan<char> AsSpan (this string? text, Range range);
```

Parameters

`text` String

The target string.

`range` Range

The range that has start and end indexes to use for slicing the string.

Returns

ReadOnlySpan<Char>

Exceptions

ArgumentNullException

`text` is `null`.

ArgumentOutOfRangeException

`range`'s start or end index is not within the bounds of the string.

ArgumentOutOfRangeException

`range`'s start index is greater than its end index.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

AsSpan(String, Int32)

Creates a new read-only span over a portion of the target string from a specified position to the end of the string.

```
C#

public static ReadOnlySpan<char> AsSpan (this string? text, int start);
```

Parameters

`text` String

The target string.

`start` Int32

The index at which to begin this slice.

Returns

ReadOnlySpan<Char>

The read-only span representation of the string.

Exceptions

ArgumentOutOfRangeException

`start` is less than 0 or greater than `text.Length`.

ArgumentNullException

Thrown when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan(String, Index)

Creates a new ReadOnlySpan<T> over a portion of the target string from a specified position to the end of the string.

```
C#

public static ReadOnlySpan<char> AsSpan (this string? text, Index
startIndex);
```

Parameters

`text` String

The target string.

`startIndex` Index

The index at which to begin this slice.

Returns

ReadOnlySpan<Char>

Exceptions

ArgumentOutOfRangeException

`startIndex` is less than 0 or greater than `text`.Length.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

AsSpan(String)

Creates a new read-only span over a string.

```
C#
public static ReadOnlySpan<char> AsSpan (this string? text);
```

Parameters

`text` String

The target string.

Returns

ReadOnlySpan<Char>

The read-only span representation of the string.

Remarks

Returns `default` when `text` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(ArraySegment<T>)

Creates a new span over a target array segment.

```
C#
public static Span<T> AsSpan<T> (this ArraySegment<T> segment);
```

Type Parameters

`T`

The type of the array segment.

Parameters

`segment` ArraySegment<T>

The array segment to convert.

Returns

Span<T>

The span representation of the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(T[])

Creates a new span over a target array.

```
C#
public static Span<T> AsSpan<T> (this T[]? array);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The array to convert.

Returns

Span<T>

The span representation of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(ArraySegment<T>, Index)

Creates a new span over a portion of the target array segment beginning at a specified index and ending at the end of the segment.

C#

```
public static Span<T> AsSpan<T> (this ArraySegment<T> segment, Index
startIndex);
```

Type Parameters

`T`

The type of the array segment.

Parameters

`segment` ArraySegment<T>

The target array segment.

`startIndex` Index

The index at which to begin the Span.

Returns

Span<T>

The span representation of the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(ArraySegment<T>, Int32)

Creates a new span over a portion of a target array segment from a specified position to the end of the segment.

C#

```
public static Span<T> AsSpan<T> (this ArraySegment<T> segment, int
start);
```

Type Parameters

`T`

The type of the array segment.

Parameters

`segment` ArraySegment<T>

The target array segment.

`start` Int32

The index at which to begin the san.

Returns

Span<T>

The span representation of the array segment.

Exceptions

ArrayTypeMismatchException

`segment` is covariant, and the array's type is not exactly `T[]`.

ArgumentOutOfRangeException

`start` is less than 0 or greater than `segment.Count`.

Remarks

Returns `default` when `segment` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(ArraySegment<T>, Range)

Creates a new span over a portion of a target array segment using the range start and end indexes.

C#

```
public static Span<T> AsSpan<T> (this ArraySegment<T> segment, Range
range);
```

Type Parameters

`T`

The type of the array segment.

Parameters

`segment` ArraySegment<T>

The target array segment.

`range` Range

The range which has start and end indexes to use for slicing the array.

Returns

Span<T>

The span representation of the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(T[], Index)

Creates a new span over the portion of the target array defined by an Index value.

C#

```
public static Span<T> AsSpan<T> (this T[]? array, Index startIndex);
```

Type Parameters

`T`

The array type.

Parameters

`array` T[]

The array to convert.

`startIndex` Index

The starting index.

Returns

Span<T>

The span representation of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(T[], Int32)

Creates a new span over a portion of the target array starting at a specified position to the end of the array.

```
C#

public static Span<T> AsSpan<T> (this T[]? array, int start);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The array to convert.

`start` Int32

The initial index from which the array will be converted.

Returns

Span<T>

The span representation of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(T[], Range)

Creates a new span over a portion of a target array defined by a Range value.

```
C#

public static Span<T> AsSpan<T> (this T[]? array, Range range);
```

Type Parameters

`T`

The type of the array.

Parameters

`array` T[]

The array to convert.

`range` Range

The range of the array to convert.

Returns

Span<T>

The span representation of the array.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(ArraySegment<T>, Int32, Int32)

Creates a new span over a portion of a target array segment from a specified position for a specified length.

C#

```
public static Span<T> AsSpan<T> (this ArraySegment<T> segment, int start,
int length);
```

Type Parameters

`T`

The type of the array segment.

Parameters

`segment` ArraySegment<T>

The target array segment.

`start` Int32

The index at which to begin the span.

`length` Int32

The number of items in the span.

Returns

Span<T>

The span representation of the array.

Exceptions

ArrayTypeMismatchException

`segment` is covariant, and the array's type is not exactly `T[]`.

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `segment`.

Remarks

Returns `default` when `segment` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

AsSpan<T>(T[], Int32, Int32)

Creates a new span over the portion of the target array beginning at a specified position for a specified length.

C#

```
public static Span<T> AsSpan<T> (this T[]? array, int start, int length);
```

Type Parameters

T

The type of the array.

Parameters

`array` T[]

The target array.

`start` Int32

The index at which to begin the span.

`length` Int32

The number of items in the span.

Returns

Span<T>

The span representation of the array.

Exceptions

ArrayTypeMismatchException

`array` is covariant, and the array's type is not exactly `T[]`".

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` is not in the range of `array`.

Remarks

Returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.BinarySearch Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

BinarySearch<T,TComparable> (ReadOnlySpan<T>, TComparable)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified `TComparable` generic type.
BinarySearch<T,TComparable>(Span<T>, TComparable)	Searches an entire sorted Span<T> for a value using the specified `TComparable` generic type.
BinarySearch<T,TComparer>(Span<T>, T, TComparer)	Searches an entire sorted Span<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T,TComparer> (ReadOnlySpan<T>, T, TComparer)	Searches an entire sorted ReadOnlySpan<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T>(ReadOnlySpan<T>, IComparable<T>)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified IComparable<T> generic interface.
BinarySearch<T>(Span<T>, IComparable<T>)	Searches an entire sorted Span<T> for a value using the specified IComparable<T> generic interface.

BinarySearch<T,TComparable> (ReadOnlySpan<T>, TComparable)

Searches an entire sorted ReadOnlySpan<T> for a value using the specified `TComparable` generic type.

```
C#
```

```
public static int BinarySearch<T,TComparable> (this ReadOnlySpan<T> span,
TComparable comparable) where TComparable : IComparable<T>;
```

Type Parameters

`T`

The element type of the span.

`TComparable`

The specific type of IComparable<T>.

Parameters

`span` ReadOnlySpan<T>

The sorted ReadOnlySpan<T> to search.

`comparable` TComparable

The `TComparable` to use when comparing.

Returns

Int32

The zero-based index of `comparable` in the sorted `span`, if `comparable` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `comparable` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparable` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

BinarySearch<T,TComparable>(Span<T>, TComparable)

Searches an entire sorted Span<T> for a value using the specified `TComparable` generic type.

C#

```
public static int BinarySearch<T,TComparable> (this Span<T> span,
TComparable comparable) where TComparable : IComparable<T>;
```

Type Parameters

`T`

The element type of the span.

`TComparable`

The specific type of IComparable<T>.

Parameters

`span` Span<T>

The sorted Span<T> to search.

`comparable` TComparable

The `TComparable` to use when comparing.

Returns

Int32

The zero-based index of `comparable` in the sorted `span`, if `comparable` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `comparable` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparable` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

BinarySearch<T,TComparer>(Span<T>, T, TComparer)

Searches an entire sorted Span<T> for a specified value using the specified `TComparer` generic type.

C#

```
public static int BinarySearch<T,TComparer> (this Span<T> span, T value,
TComparer comparer) where TComparer :
System.Collections.Generic.IComparer<T>;
```

Type Parameters

`T`

The element type of the span.

`TComparer`

The specific type of IComparer<T>.

Parameters

`span` Span<T>

The sorted Span<T> to search.

`value` T

The object to locate. The value can be `null` for reference types.

`comparer` TComparer

The `TComparer` to use when comparing.

Returns

Int32

The zero-based index of `value` in the sorted `span`, if `value` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `value` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparer` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

BinarySearch<T,TComparer> (ReadOnlySpan<T>, T, TComparer)

Searches an entire sorted ReadOnlySpan<T> for a specified value using the specified `TComparer` generic type.

C#

```
public static int BinarySearch<T,TComparer> (this ReadOnlySpan<T> span, T
value, TComparer comparer) where TComparer :
System.Collections.Generic.IComparer<T>;
```

Type Parameters

`T`

The element type of the span.

`TComparer`

The specific type of IComparer<T>.

Parameters

`span` ReadOnlySpan<T>

The sorted ReadOnlySpan<T> to search.

`value` T

The object to locate. The value can be `null` for reference types.

`comparer` TComparer

The `TComparer` to use when comparing.

Returns

Int32

The zero-based index of `value` in the sorted `span`, if `value` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `value` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparer` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

BinarySearch<T>(ReadOnlySpan<T>, IComparable<T>)

Searches an entire sorted ReadOnlySpan<T> for a value using the specified IComparable<T> generic interface.

```
C#

public static int BinarySearch<T> (this ReadOnlySpan<T> span,
IComparable<T> comparable);
```

Type Parameters

T

The element type of the span.

Parameters

span ReadOnlySpan<T>

The sorted ReadOnlySpan<T> to search.

comparable IComparable<T>

The IComparable<T> to use when comparing.

Returns

Int32

The zero-based index of `comparable` in the sorted `span`, if `comparable` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `comparable` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparable` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

BinarySearch<T>(Span<T>, IComparable<T>)

Searches an entire sorted Span<T> for a value using the specified IComparable<T> generic interface.

C#

```
public static int BinarySearch<T> (this Span<T> span, IComparable<T>
comparable);
```

Type Parameters

T

The element type of the span.

Parameters

span Span<T>

The sorted Span<T> to search.

comparable IComparable<T>

The IComparable<T> to use when comparing.

Returns

Int32

The zero-based index of `comparable` in the sorted `span`, if `comparable` is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `comparable` or, if there is no larger element, the bitwise complement of Length.

Exceptions

ArgumentNullException

`comparable` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.CommonPrefixLength Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Finds the length of any common prefix shared between `span` and `other`.

CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Finds the length of any common prefix shared between `span` and `other`.

C#

```
public static int CommonPrefixLength<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other);
```

Type Parameters

T

The type of the elements in the spans.

Parameters

`span` ReadOnlySpan<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

Returns

Int32

The length of the common prefix shared by the two spans. If there's no shared prefix, 0 is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>)

Finds the length of any common prefix shared between `span` and `other`.

C#

```
public static int CommonPrefixLength<T> (this Span<T> span,
ReadOnlySpan<T> other);
```

Type Parameters

`T`

The type of the elements in the spans.

Parameters

`span` Span<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

Returns

Int32

The length of the common prefix shared by the two spans. If there's no shared prefix, 0 is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)

Determines the length of any common prefix shared between `span` and `other`.

C#

```
public static int CommonPrefixLength<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other, System.Collections.Generic.IEqualityComparer<T>?
comparer);
```

Type Parameters

`T`

The type of the elements in the sequences.

Parameters

`span` ReadOnlySpan<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing elements, or `null` to use the default IEqualityComparer<T> for the type of an element.

Returns

Int32

The length of the common prefix shared by the two spans. If there's no shared prefix, 0 is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)

Finds the length of any common prefix shared between `span` and `other`.

C#

```
public static int CommonPrefixLength<T> (this Span<T> span,
ReadOnlySpan<T> other, System.Collections.Generic.IEqualityComparer<T>?
comparer);
```

Type Parameters

`T`

The type of the elements in the spans.

Parameters

`span` Span<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing elements, or `null` to use the default IEqualityComparer<T> for the type of an element.

Returns

Int32

The length of the common prefix shared by the two spans. If there's no shared prefix, 0 is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Compares one character span with another using a specified string comparison, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int CompareTo (this ReadOnlySpan<char> span,
ReadOnlySpan<char> other, StringComparison comparisonType);
```

Parameters

span ReadOnlySpan<Char>

The source span.

other ReadOnlySpan<Char>

The value to compare with the source span.

comparisonType StringComparison

An enumeration value that determines how `span` and `other` are compared.

Returns

Int32

A signed integer that indicates the relative order of `span` and `other`:
- If less than 0, `span` precedes than `other`.
- If 0, `span` equals `other`.
- If greater than 0, `span` follows `other`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Contains Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Contains(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Indicates whether a specified value occurs within a read-only character span.
Contains<T>(ReadOnlySpan<T>, T)	Indicates whether a specified value is found in a read-only span. Values are compared using IEquatable{T}.Equals(T).
Contains<T>(Span<T>, T)	Indicates whether a specified value is found in a span. Values are compared using IEquatable{T}.Equals(T).

Contains(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)

Indicates whether a specified value occurs within a read-only character span.

C#

```
public static bool Contains (this ReadOnlySpan<char> span,
ReadOnlySpan<char> value, StringComparison comparisonType);
```

Parameters

span ReadOnlySpan<Char>

The source span.

value ReadOnlySpan<Char>

The value to seek within the source span.

`comparisonType` StringComparison

An enumeration value that determines how the characters in `span` and `value` are compared.

Returns

Boolean

`true` if `value` occurs within the span, `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Contains<T>(ReadOnlySpan<T>, T)

Indicates whether a specified value is found in a read-only span. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static bool Contains<T> (this ReadOnlySpan<T> span, T value) where
T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

The value to search for.

Returns

Boolean

`true` if found, `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Contains<T>(Span<T>, T)

Indicates whether a specified value is found in a span. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static bool Contains<T> (this Span<T> span, T value) where T :
IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The span to search.

`value` T

The value to search for.

Returns

Boolean

`true` if found, `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ContainsAny Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

ContainsAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.

ContainsAny<T>(ReadOnlySpan<T>, T, T, T)

Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.

```
C#
```

```
public static bool ContainsAny<T> (this ReadOnlySpan<T> span, T value0, T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

`value2` T

One of the values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(Span<T>, T, T)

Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.

```
C#
```

```
public static bool ContainsAny<T> (this Span<T> span, T value0, T value1)
where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(ReadOnlySpan<T>, T, T)

Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAny<T> (this ReadOnlySpan<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAny<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(Span<T>, SearchValues<T>)

Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAny<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAny<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` ReadOnlySpan<T>

The set of values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(Span<T>, T, T, T)

Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.

```
C#

public static bool ContainsAny<T> (this Span<T> span, T value0, T value1,
T value2) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span Span<T>

The span to search.

value0 T

One of the values to search for.

value1 T

One of the values to search for.

value2 T

One of the values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAny<T>(Span<T>, ReadOnlySpan<T>)

Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.

```
C#

public static bool ContainsAny<T> (this Span<T> span, ReadOnlySpan<T>
values) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span Span<T>

The span to search.

values ReadOnlySpan<T>

The set of values to search for.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.ContainsAnyExcept Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

ContainsAnyExcept<T>(Span<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(Span<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExcept<T>(Span<T>, T)	Searches for any value other than the specified `value`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for any value other than the specified `value`.

ContainsAnyExcept<T>(Span<T>, T, T)

Searches for any value other than `value0` or `value1`.

```
C#

public static bool ContainsAnyExcept<T> (this Span<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`value0` T

The value to exclude from the search.

`value1` T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value0` or `value1` is present in the span. If all of the values are `value0` or `value1`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T)

Searches for any value other than `value0` or `value1`.

```
C#

public static bool ContainsAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span ReadOnlySpan<T>

The span to search.

value0 T

The value to exclude from the search.

value1 T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value0` or `value1` is present in the span. If all of the values are `value0` or `value1`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(Span<T>, T, T, T)

Searches for any value other than `value0`, `value1`, or `value2`.

```
C#

public static bool ContainsAnyExcept<T> (this Span<T> span, T value0, T
value1, T value2) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span Span<T>

The span to search.

value0 T

The value to exclude from the search.

value1 T

The value to exclude from the search.

value2 T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value0`, `value1`, or `value2` is present in the span. If all of the values are `value0`, `value1`, or `value2`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T, T)

Searches for any value other than `value0`, `value1`, or `value2`.

```
C#

public static bool ContainsAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

The value to exclude from the search.

`value1` T

The value to exclude from the search.

`value2` T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value0`, `value1`, or `value2` is present in the span. If all of the values are `value0`, `value1`, or `value2`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(Span<T>, T)

Searches for any value other than the specified `value`.

```
C#

public static bool ContainsAnyExcept<T> (this Span<T> span, T value)
where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span Span<T>

The span to search.

value T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value` is present in the span. If all of the values are `value`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for any value other than the specified `values`.

```
C#

public static bool ContainsAnyExcept<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span ReadOnlySpan<T>

The span to search.

values ReadOnlySpan<T>

The set of values to exclude from the search.

Returns

Boolean

`true` if any value other than those in `values` is present in the span. If all of the values are in `values`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(Span<T>, SearchValues<T>)

Searches for any value other than the specified `values`.

```
C#

public static bool ContainsAnyExcept<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

T

Parameters

span Span<T>

The span to search.

values SearchValues<T>

The set of values to exclude from the search.

Returns

Boolean

`true` if any value other than those in `values` is present in the span. If all of the values are in `values`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for any value other than the specified `values`.

```
C#
```

```
public static bool ContainsAnyExcept<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` SearchValues<T>

The set of values to exclude from the search.

Returns

Boolean

`true` if any value other than those in `values` is present in the span. If all of the values are in `values`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(Span<T>, ReadOnlySpan<T>)

Searches for any value other than the specified `values`.

C#

```
public static bool ContainsAnyExcept<T> (this Span<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`values` ReadOnlySpan<T>

The set of values to exclude from the search.

Returns

Boolean

`true` if any value other than those in `values` is present in the span. If all of the values are in `values`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExcept<T>(ReadOnlySpan<T>, T)

Searches for any value other than the specified `value`.

```
C#

public static bool ContainsAnyExcept<T> (this ReadOnlySpan<T> span, T
value) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

The value to exclude from the search.

Returns

Boolean

`true` if any value other than `value` is present in the span. If all of the values are `value`, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ContainsAnyExceptInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

ContainsAnyExceptInRange<T> (ReadOnlySpan<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
ContainsAnyExceptInRange<T>(Span<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

ContainsAnyExceptInRange<T> (ReadOnlySpan<T>, T, T)

Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static bool ContainsAnyExceptInRange<T> (this ReadOnlySpan<T>
span, T lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

Parameters

span ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Boolean

`true` if any value other than those in the specified range is present in the span. If all of the values are inside of the specified range, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyExceptInRange<T>(Span<T>, T, T)

Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static bool ContainsAnyExceptInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Boolean

`true` if any value other than those in the specified range is present in the span. If all of the values are inside of the specified range, returns `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ContainsAnyInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

ContainsAnyInRange<T> (ReadOnlySpan<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.
ContainsAnyInRange<T> (Span<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.

ContainsAnyInRange<T>(ReadOnlySpan<T>, T, T)

Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAnyInRange<T> (this ReadOnlySpan<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

Parameters

`span` ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ContainsAnyInRange<T>(Span<T>, T, T)

Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.

C#

```
public static bool ContainsAnyInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Boolean

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.CopyTo Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

CopyTo<T>(T[], Memory<T>)	Copies the contents of the array into a memory region.
CopyTo<T>(T[], Span<T>)	Copies the contents of the array into the span.

CopyTo<T>(T[], Memory<T>)

Copies the contents of the array into a memory region.

C#

```
public static void CopyTo<T> (this T[]? source, Memory<T> destination);
```

Type Parameters

`T`

The type of the array.

Parameters

`source` T[]

The array to copy items from.

`destination` Memory<T>

The memory to copy items into.

Exceptions

ArgumentException

The destination is shorter than the source array.

Remarks

If `source` and `destination` overlap, this method behaves as if the original values are in a temporary location before the destination is overwritten.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

CopyTo<T>(T[], Span<T>)

Copies the contents of the array into the span.

C#

```
public static void CopyTo<T> (this T[]? source, Span<T> destination);
```

Type Parameters

T

The type of the array.

Parameters

source T[]

The array to copy items from.

destination Span<T>

The span to copy items into.

Exceptions

ArgumentException

The destination Span is shorter than the source array.

Remarks

If `source` and `destination` overlap, this method behaves as if the original values are in a temporary location before the destination is overwritten.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Count Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Count<T>(ReadOnlySpan<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(Span<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Counts the number of times `value` occurs in `span`.
Count<T>(Span<T>, ReadOnlySpan<T>)	Counts the number of times the specified `value` occurs in the `span`.

Count<T>(ReadOnlySpan<T>, T)

Counts the number of times the specified `value` occurs in the `span`.

C#

```
public static int Count<T> (this ReadOnlySpan<T> span, T value) where T :
IEquatable<T>;
```

Type Parameters

T

The element type of the span.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

The value for which to search.

Returns

Int32

The number of times `value` was found in the `span`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Count<T>(Span<T>, T)

Counts the number of times the specified `value` occurs in the `span`.

C#

```
public static int Count<T> (this Span<T> span, T value) where T :
IEquatable<T>;
```

Type Parameters

`T`

The element type of the span.

Parameters

`span` Span<T>

The span to search.

`value` T

The value for which to search.

Returns

Int32

The number of times `value` was found in the `span`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Count<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Counts the number of times `value` occurs in `span`.

C#

```
public static int Count<T> (this ReadOnlySpan<T> span, ReadOnlySpan<T>
value) where T : IEquatable<T>;
```

Type Parameters

T

The element type of the span.

Parameters

span ReadOnlySpan<T>

The span to search.

value ReadOnlySpan<T>

The value for which to search.

Returns

Int32

The number of times `value` was found in `span`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Count<T>(Span<T>, ReadOnlySpan<T>)

Counts the number of times the specified `value` occurs in the `span`.

C#

```
public static int Count<T> (this Span<T> span, ReadOnlySpan<T> value)
where T : IEquatable<T>;
```

Type Parameters

`T`

The element type of the span.

Parameters

`span` Span<T>

The span to search.

`value` ReadOnlySpan<T>

The value for which to search.

Returns

Int32

The number of times `value` was found in the `span`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.EndsWith Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

EndsWith(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Determines whether the end of the `span` matches the specified `value` when compared using the specified `comparisonType` option.
EndsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a read-only span.
EndsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a span.

EndsWith(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)

Determines whether the end of the `span` matches the specified `value` when compared using the specified `comparisonType` option.

C#

```
public static bool EndsWith (this ReadOnlySpan<char> span,
ReadOnlySpan<char> value, StringComparison comparisonType);
```

Parameters

span ReadOnlySpan<Char>

The source span.

value ReadOnlySpan<Char>

The sequence to compare to the end of the source span.

`comparisonType` StringComparison

An enumeration value that determines how `span` and `value` are compared.

Returns

Boolean

`true` if `value` matches the end of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

EndsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Determines whether the specified sequence appears at the end of a read-only span.

C#

```
public static bool EndsWith<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span.

Parameters

`span` ReadOnlySpan<T>

The source span.

`value` ReadOnlySpan<T>

The sequence to compare to the end of the source span.

Returns

Boolean

`true` if `value` matches the end of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

EndsWith<T>(Span<T>, ReadOnlySpan<T>)

Determines whether the specified sequence appears at the end of a span.

C#

```
public static bool EndsWith<T> (this Span<T> span, ReadOnlySpan<T> value)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span.

Parameters

`span` Span<T>

The source span.

`value` ReadOnlySpan<T>

The sequence to compare to the end of the source span.

Returns

Boolean

`true` if `value` matches the end of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.EnumerateLines Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

EnumerateLines(ReadOnlySpan<Char>)	Returns an enumeration of lines over the provided span.
EnumerateLines(Span<Char>)	Returns an enumeration of lines over the provided span.

EnumerateLines(ReadOnlySpan<Char>)

Returns an enumeration of lines over the provided span.

C#

```
public static System.Text.SpanLineEnumerator EnumerateLines (this
ReadOnlySpan<char> span);
```

Parameters

span ReadOnlySpan<Char>

A span containing the lines to enumerate.

Returns

SpanLineEnumerator

An enumeration of lines.

Remarks

It is recommended that protocol parsers not utilize this API. See the documentation for ReplaceLineEndings() for more information on how newline sequences are detected.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

EnumerateLines(Span<Char>)

Returns an enumeration of lines over the provided span.

C#

```
public static System.Text.SpanLineEnumerator EnumerateLines (this
Span<char> span);
```

Parameters

span Span<Char>

A span containing the lines to enumerate.

Returns

SpanLineEnumerator

An enumeration of lines.

Remarks

It is recommended that protocol parsers not utilize this API. See the documentation for ReplaceLineEndings() for more information on how newline sequences are detected.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.EnumerateRunes Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

EnumerateRunes(ReadOnlySpan<Char>)	Returns an enumeration of Rune from the provided read-only span.
EnumerateRunes(Span<Char>)	Returns an enumeration of Rune from the provided span.

EnumerateRunes(ReadOnlySpan<Char>)

Returns an enumeration of Rune from the provided read-only span.

C#

```
public static System.Text.SpanRuneEnumerator EnumerateRunes (this
ReadOnlySpan<char> span);
```

Parameters

span ReadOnlySpan<Char>

The source span.

Returns

SpanRuneEnumerator

A rune enumerator.

Remarks

Invalid sequences will be represented in the enumeration by Rune.ReplacementChar.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

EnumerateRunes(Span<Char>)

Returns an enumeration of Rune from the provided span.

C#

```
public static System.Text.SpanRuneEnumerator EnumerateRunes (this
Span<char> span);
```

Parameters

span Span<Char>

The source span.

Returns

SpanRuneEnumerator

A rune enumerator.

Remarks

Invalid sequences will be represented in the enumeration by ReplacementChar.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Determines whether this `span` and the specified `other` span have the same characters when compared using the specified `comparisonType` option.

C#

```
public static bool Equals (this ReadOnlySpan<char> span, ReadOnlySpan<char>
other, StringComparison comparisonType);
```

Parameters

`span` ReadOnlySpan<Char>

The source span.

`other` ReadOnlySpan<Char>

The value to compare with the source span.

`comparisonType` StringComparison

An enumeration value that determines how `span` and `other` are compared.

Returns

Boolean

`true` if equal, `false` otherwise.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MemoryExtensions.IndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

IndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Reports the zero-based index of the first occurrence of the specified `value` in the current `span`.
IndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).

IndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)

Reports the zero-based index of the first occurrence of the specified `value` in the current `span`.

C#

```
public static int IndexOf (this ReadOnlySpan<char> span,
ReadOnlySpan<char> value, StringComparison comparisonType);
```

Parameters

`span` ReadOnlySpan<Char>

The source span.

`value` ReadOnlySpan<Char>

The value to seek within the source span.

`comparisonType` StringComparison

An enumeration value that determines how `span` and `value` are compared.

Returns

Int32

The index of the occurrence of the value in the span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOf<T>(Span<T>, ReadOnlySpan<T>)

Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int IndexOf<T> (this Span<T> span, ReadOnlySpan<T> value)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` Span<T>

The span to search.

`value` ReadOnlySpan<T>

The sequence to search for.

Returns

Int32

The index of the occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOf<T>(Span<T>, T)

Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int IndexOf<T> (this Span<T> span, T value) where T :
IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` Span<T>

The span to search.

`value` T

The value to search for.

Returns

Int32

The index of the occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOf<T>(ReadOnlySpan<T>, T)

Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int IndexOf<T> (this ReadOnlySpan<T> span, T value) where T
: IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

The value to search for.

Returns

Int32

The index of the occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int IndexOf<T> (this ReadOnlySpan<T> span, ReadOnlySpan<T>
value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` ReadOnlySpan<T>

The sequence to search for.

Returns

Int32

The index of the occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.IndexOfAny Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

IndexOfAny<T>(Span<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

IndexOfAny<T>(Span<T>, T, T, T)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

```
C#

public static int IndexOfAny<T> (this Span<T> span, T value0, T value1, T
value2) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

value0 T

One of the values to search for.

value1 T

One of the values to search for.

value2 T

One of the values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

IndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for the first index of any of the specified values.

C#

```
public static int IndexOfAny<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Int32

The first index of any of the specified values, or -1 if none are found.

Remarks

This method is similar to calling IndexOf several times with the logical OR operator.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

C#

```
public static int IndexOfAny<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

values ReadOnlySpan<T>

The set of values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOfAny<T>(Span<T>, SearchValues<T>)

Searches for the first index of any of the specified values.

C#

```
public static int IndexOfAny<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Int32

The first index of any of the specified values, or -1 if none are found.

Remarks

This method is similar to calling IndexOf several times with the logical OR operator.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAny<T>(Span<T>, ReadOnlySpan<T>)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

```
C#

public static int IndexOfAny<T> (this Span<T> span, ReadOnlySpan<T>
values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

values ReadOnlySpan<T>

The set of values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

IndexOfAny<T>(ReadOnlySpan<T>, T, T)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

```
C#

public static int IndexOfAny<T> (this ReadOnlySpan<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

value0 T

One of the values to search for.

value1 T

One of the values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOfAny<T>(Span<T>, T, T)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

```
C#

public static int IndexOfAny<T> (this Span<T> span, T value0, T value1)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOfAny<T>(ReadOnlySpan<T>, T, T, T)

Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.

C#

```
public static int IndexOfAny<T> (this ReadOnlySpan<T> span, T value0, T
value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

`value2` T

One of the values to search for.

Returns

Int32

The first index of the occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.IndexOfAnyExcept Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

IndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(Span<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.
IndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.

IndexOfAnyExcept<T>(Span<T>, T, T)

Searches for the first index of any value other than the specified `value0` or `value1`.

```
C#

public static int IndexOfAnyExcept<T> (this Span<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

value0 T

A value to avoid.

value1 T

A value to avoid

Returns

Int32

The index in the span of the first occurrence of any value other than `value0` and `value1`. If all of the values are `value0` or `value1`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)

Searches for the first index of any value other than the specified `value0` or `value1`.

```
C#

public static int IndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

value0 T

A value to avoid.

value1 T

A value to avoid

Returns

Int32

The index in the span of the first occurrence of any value other than `value0` and `value1`. If all of the values are `value0` or `value1`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(Span<T>, T)

Searches for the first index of any value other than the specified `value`.

```
C#

public static int IndexOfAnyExcept<T> (this Span<T> span, T value) where
T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

value T

A value to avoid.

Returns

Int32

The index in the span of the first occurrence of any value other than `value`. If all of the values are `value`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)

Searches for the first index of any value other than the specified `values`.

```
C#
```

```
public static int IndexOfAnyExcept<T> (this Span<T> span, ReadOnlySpan<T>
values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

values ReadOnlySpan<T>

The values to avoid.

Returns

Int32

The index in the span of the first occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for the first index of any value other than the specified `values`.

C#

```
public static int IndexOfAnyExcept<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` SearchValues<T>

The values to exclude from the search.

Returns

Int32

The index in the span of the first occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAnyExcept<T>(ReadOnlySpan<T>, T)

Searches for the first index of any value other than the specified `value`.

C#

```
public static int IndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

A value to avoid.

Returns

Int32

The index in the span of the first occurrence of any value other than `value`. If all of the values are `value`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the first index of any value other than the specified `values`.

C#

```
public static int IndexOfAnyExcept<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` ReadOnlySpan<T>

The values to avoid.

Returns

Int32

The index in the span of the first occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)

Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.

C#

```
public static int IndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

`value2` T

A value to avoid

Returns

Int32

The index in the span of the first occurrence of any value other than `value0`, `value1`, and `value2`. If all of the values are `value0`, `value1`, and `value2`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

IndexOfAnyExcept<T>(Span<T>, SearchValues<T>)

Searches for the first index of any value other than the specified `values`.

C#

```
public static int IndexOfAnyExcept<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`values` SearchValues<T>

The values to exclude from the search.

Returns

Int32

The index in the span of the first occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAnyExcept<T>(Span<T>, T, T, T)

Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.

```
C#

public static int IndexOfAnyExcept<T> (this Span<T> span, T value0, T
value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

`value2` T

A value to avoid

Returns

Int32

The index in the span of the first occurrence of any value other than `value0`, `value1`, and `value2`. If all of the values are `value0`, `value1`, and `value2`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.IndexOfAnyExceptInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

IndexOfAnyExceptInRange<T> (Span<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyExceptInRange<T> (ReadOnlySpan<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

IndexOfAnyExceptInRange<T>(Span<T>, T, T)

Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int IndexOfAnyExceptInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Int32

The index in the span of the first occurrence of any value outside of the specified range. If all of the values are inside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAnyExceptInRange<T> (ReadOnlySpan<T>, T, T)

Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int IndexOfAnyExceptInRange<T> (this ReadOnlySpan<T> span,
T lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Int32

The index in the span of the first occurrence of any value outside of the specified range. If all of the values are inside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.IndexOfAnyInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

IndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

IndexOfAnyInRange<T>(Span<T>, T, T)

Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int IndexOfAnyInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Int32

The index in the span of the first occurrence of any value in the specified range. If all of the values are outside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

IndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)

Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

```
C#

public static int IndexOfAnyInRange<T> (this ReadOnlySpan<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Int32

The index in the span of the first occurrence of any value in the specified range. If all of the values are outside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.IsWhiteSpace(ReadOnlySpan<Char>) Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Indicates whether the specified span contains only whitespace characters.

C#

```
public static bool IsWhiteSpace (this ReadOnlySpan<char> span);
```

Parameters

span ReadOnlySpan<Char>

The source span.

Returns

Boolean

`true` if the span contains only whitespace characters, `false` otherwise.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

MemoryExtensions.LastIndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

LastIndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Reports the zero-based index of the last occurrence of the specified `value` in the current `span`.
LastIndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

LastIndexOf(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)

Reports the zero-based index of the last occurrence of the specified `value` in the current `span`.

C#

```
public static int LastIndexOf (this ReadOnlySpan<char> span,
```

```
ReadOnlySpan<char> value, StringComparison comparisonType);
```

Parameters

span ReadOnlySpan<Char>

The source span.

value ReadOnlySpan<Char>

The value to seek within the source span.

comparisonType StringComparison

An enumeration value that determines how `span` and `value` are compared.

Returns

Int32

The index of the last occurrence of the value in the span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

LastIndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int LastIndexOf<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` ReadOnlySpan<T>

The sequence to search for.

Returns

Int32

The index of the last occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOf<T>(ReadOnlySpan<T>, T)

Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int LastIndexOf<T> (this ReadOnlySpan<T> span, T value)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value` T

The value to search for.

Returns

Int32

The index of the last occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOf<T>(Span<T>, ReadOnlySpan<T>)

Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int LastIndexOf<T> (this Span<T> span, ReadOnlySpan<T>
value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` Span<T>

The span to search.

`value` ReadOnlySpan<T>

The sequence to search for.

Returns

Int32

The index of the last occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOf<T>(Span<T>, T)

Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

C#

```
public static int LastIndexOf<T> (this Span<T> span, T value) where T :
IEquatable<T>;
```

Type Parameters

`T`

The type of the span and value.

Parameters

`span` Span<T>

The span to search.

`value` T

The value to search for.

Returns

Int32

The index of the last occurrence of the value in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.LastIndexOfAny Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

LastIndexOfAny<T>(Span<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

LastIndexOfAny<T>(Span<T>, T, T, T)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

```
C#

public static int LastIndexOfAny<T> (this Span<T> span, T value0, T
value1, T value2) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

value0 T

One of the values to search for.

value1 T

One of the values to search for.

value2 T

One of the values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOfAny<T>(ReadOnlySpan<T>, T, T, T)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

```
C#

public static int LastIndexOfAny<T> (this ReadOnlySpan<T> span, T value0,
T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

`value2` T

One of the values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOfAny<T>(ReadOnlySpan<T>, T, T)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

C#

```
public static int LastIndexOfAny<T> (this ReadOnlySpan<T> span, T value0,
T value1) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOfAny<T>(Span<T>, T, T)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

C#

```
public static int LastIndexOfAny<T> (this Span<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value0` T

One of the values to search for.

`value1` T

One of the values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOfAny<T>(Span<T>, SearchValues<T>)

Searches for the last index of any of the specified values.

```
C#

public static int LastIndexOfAny<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` Span<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Int32

The last index of any of the specified values, or -1 if none are found.

Remarks

This method is similar to calling LastIndexOf several times with the logical OR operator.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

C#

```
public static int LastIndexOfAny<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` ReadOnlySpan<T>

The set of values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

LastIndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for the last index of any of the specified values.

C#

```
public static int LastIndexOfAny<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

Parameters

`span` ReadOnlySpan<T>

The span to search.

`values` SearchValues<T>

The set of values to search for.

Returns

Int32

The last index of any of the specified values, or -1 if none are found.

Remarks

This method is similar to calling LastIndexOf several times with the logical OR operator.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAny<T>(Span<T>, ReadOnlySpan<T>)

Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.

C#

```
public static int LastIndexOfAny<T> (this Span<T> span, ReadOnlySpan<T>
values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span Span<T>

The span to search.

values ReadOnlySpan<T>

The set of values to search for.

Returns

Int32

The index of the last occurrence of any of the values in the span. If not found, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.LastIndexOfAnyExcept Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(Span<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)

Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.

```
C#

public static int LastIndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

`value2` T

A value to avoid

Returns

Int32

The index in the span of the last occurrence of any value other than `value0`, `value1`, and `value2`. If all of the values are `value0`, `value1`, and `value2`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)

Searches for the last index of any value other than the specified `values`.

```
C#

public static int LastIndexOfAnyExcept<T> (this ReadOnlySpan<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

values SearchValues<T>

The values to exclude from the search.

Returns

Int32

The index in the span of the last occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Searches for the last index of any value other than the specified `values`.

```
C#

public static int LastIndexOfAnyExcept<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

values ReadOnlySpan<T>

The values to avoid.

Returns

Int32

The index in the span of the first occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T)

Searches for the last index of any value other than the specified `value`.

```
C#

public static int LastIndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value) where T : IEquatable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

value T

A value to avoid.

Returns

Int32

The index in the span of the last occurrence of any value other than `value`. If all of the values are `value`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(Span<T>, SearchValues<T>)

Searches for the last index of any value other than the specified `values`.

```
C#
```

```
public static int LastIndexOfAnyExcept<T> (this Span<T> span,
System.Buffers.SearchValues<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`values` SearchValues<T>

The values to exclude from the search.

Returns

Int32

The index in the span of the last occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)

Searches for the last index of any value other than the specified `values`.

C#

```
public static int LastIndexOfAnyExcept<T> (this Span<T> span,
ReadOnlySpan<T> values) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`values` ReadOnlySpan<T>

The values to avoid.

Returns

Int32

The index in the span of the last occurrence of any value other than those in `values`. If all of the values are in `values`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(Span<T>, T)

Searches for the last index of any value other than the specified `value`.

C#

```
public static int LastIndexOfAnyExcept<T> (this Span<T> span, T value)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value` T

A value to avoid.

Returns

Int32

The index in the span of the last occurrence of any value other than `value`. If all of the values are `value`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)

Searches for the last index of any value other than the specified `value0` or `value1`.

C#

```
public static int LastIndexOfAnyExcept<T> (this ReadOnlySpan<T> span, T
value0, T value1) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

Returns

Int32

The index in the span of the last occurrence of any value other than `value0` and `value1`. If all of the values are `value0` or `value1`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(Span<T>, T, T)

Searches for the last index of any value other than the specified `value0` or `value1`.

C#

```
public static int LastIndexOfAnyExcept<T> (this Span<T> span, T value0, T
value1) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

Returns

Int32

The index in the span of the last occurrence of any value other than `value0` and `value1`. If all of the values are `value0` or `value1`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

LastIndexOfAnyExcept<T>(Span<T>, T, T, T)

Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.

C#

```
public static int LastIndexOfAnyExcept<T> (this Span<T> span, T value0, T value1, T value2) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`value0` T

A value to avoid.

`value1` T

A value to avoid

`value2` T

A value to avoid

Returns

Int32

The index in the span of the last occurrence of any value other than `value0`, `value1`, and `value2`. If all of the values are `value0`, `value1`, and `value2`, returns -1.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.LastIndexOfAnyExceptInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

LastIndexOfAnyExceptInRange<T>(Span<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

LastIndexOfAnyExceptInRange<T>(Span<T>, T, T)

Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int LastIndexOfAnyExceptInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Int32

The index in the span of the last occurrence of any value outside of the specified range. If all of the values are inside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAnyExceptInRange<T> (ReadOnlySpan<T>, T, T)

Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int LastIndexOfAnyExceptInRange<T> (this ReadOnlySpan<T>
span, T lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the excluded range.

`highInclusive` T

The upper bound, inclusive, of the excluded range.

Returns

Int32

The index in the span of the last occurrence of any value outside of the specified range. If all of the values are inside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.LastIndexOfAnyInRange Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

LastIndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

LastIndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)

Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

C#

```
public static int LastIndexOfAnyInRange<T> (this ReadOnlySpan<T> span, T lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

T

The type of the span and values.

Parameters

span ReadOnlySpan<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Int32

The index in the span of the last occurrence of any value in the specified range. If all of the values are outside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

LastIndexOfAnyInRange<T>(Span<T>, T, T)

Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

```
C#

public static int LastIndexOfAnyInRange<T> (this Span<T> span, T
lowInclusive, T highInclusive) where T : IComparable<T>;
```

Type Parameters

`T`

The type of the span and values.

Parameters

`span` Span<T>

The span to search.

`lowInclusive` T

The lower bound, inclusive, of the range for which to search.

`highInclusive` T

The upper bound, inclusive, of the range for which to search.

Returns

Int32

The index in the span of the last occurrence of any value in the specified range. If all of the values are outside of the specified range, returns -1.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Overlaps Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences overlap in memory.
Overlaps<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span overlap in memory.
Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, Int32)	Determines whether two read-only sequences overlap in memory and outputs the element offset.
Overlaps<T>(Span<T>, ReadOnlySpan<T>, Int32)	Determines whether a span and a read-only span overlap in memory and outputs the element offset.

Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Determines whether two read-only sequences overlap in memory.

C#

```
public static bool Overlaps<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other);
```

Type Parameters

T

The type of elements in the read-only sequence.

Parameters

`span` ReadOnlySpan<T>

The first sequence.

`other` ReadOnlySpan<T>

The second sequence.

Returns

Boolean

`true` if the two sequences overlap; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Overlaps<T>(Span<T>, ReadOnlySpan<T>)

Determines whether a span and a read-only span overlap in memory.

C#

```
public static bool Overlaps<T> (this Span<T> span, ReadOnlySpan<T>
other);
```

Type Parameters

`T`

The type of elements in the span.

Parameters

`span` Span<T>

The span to compare.

`other` ReadOnlySpan<T>

The read-only span to compare.

Returns

Boolean

`true` if the two sequences overlap; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, Int32)

Determines whether two read-only sequences overlap in memory and outputs the element offset.

C#

```
public static bool Overlaps<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other, out int elementOffset);
```

Type Parameters

`T`

The type of elements in the span.

Parameters

`span` ReadOnlySpan<T>

The first sequence.

`other` ReadOnlySpan<T>

The second sequence.

`elementOffset` Int32

When the method returns, contains the offset between `span` and `other`.

Returns

Boolean

`true` if the two sequences overlap; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Overlaps<T>(Span<T>, ReadOnlySpan<T>, Int32)

Determines whether a span and a read-only span overlap in memory and outputs the element offset.

C#

```
public static bool Overlaps<T> (this Span<T> span, ReadOnlySpan<T> other,
out int elementOffset);
```

Type Parameters

`T`

The type of elements in the span.

Parameters

`span` Span<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

`elementOffset` Int32

When the method returns, contains the offset between `span` and `other`.

Returns

Boolean

`true` if the two sequences overlap; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Replace Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Replace<T>(Span<T>, T, T)	Replaces all occurrences of `oldValue` with `newValue`.
Replace<T>(ReadOnlySpan<T>, Span<T>, T, T)	Copies `source` to `destination`, replacing all occurrences of `oldValue` with `newValue`.

Replace<T>(Span<T>, T, T)

Replaces all occurrences of `oldValue` with `newValue`.

C#

```
public static void Replace<T> (this Span<T> span, T oldValue, T newValue)
where T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the span.

Parameters

span Span<T>

The span in which the elements should be replaced.

oldValue T

The value to be replaced with `newValue`.

`newValue` T

The value to replace all occurrences of `oldValue`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Replace<T>(ReadOnlySpan<T>, Span<T>, T, T)

Copies `source` to `destination`, replacing all occurrences of `oldValue` with `newValue`.

```
C#

public static void Replace<T> (this ReadOnlySpan<T> source, Span<T>
destination, T oldValue, T newValue) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the spans.

Parameters

`source` ReadOnlySpan<T>

The span to copy.

`destination` Span<T>

The span into which the copied and replaced values should be written.

`oldValue` T

The value to be replaced with `newValue`.

`newValue` T

The value to replace all occurrences of `oldValue`.

Exceptions

ArgumentException

The `destination` span is shorter than the `source` span. -or- `source` and `destination` overlap but don't refer to the same starting location.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Reverse<T> (Span<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Reverses the sequence of the elements in the entire span.

C#

```
public static void Reverse<T> (this Span<T> span);
```

Type Parameters

T

The type of elements in the span.

Parameters

span Span<T>

The span to reverse.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

MemoryExtensions.SequenceCompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

SequenceCompareTo<T>(Span<T>, ReadOnlySpan<T>)	Determines the relative order of a span and a read-only span by comparing the elements using IComparable{T}.CompareTo(T).
SequenceCompareTo<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines the relative order of two read-only sequences by comparing their elements using IComparable{T}.CompareTo(T).

SequenceCompareTo<T>(Span<T>, ReadOnlySpan<T>)

Determines the relative order of a span and a read-only span by comparing the elements using IComparable{T}.CompareTo(T).

C#

```
public static int SequenceCompareTo<T> (this Span<T> span,
ReadOnlySpan<T> other) where T : IComparable<T>;
```

Type Parameters

T

The type of elements in the span.

Parameters

`span` Span<T>

The span to compare.

`other` ReadOnlySpan<T>

The read-only span to compare.

Returns

Int32

A signed integer that indicates the relative order of `span` and `other`:
- If less than 0, `span` precedes than `other`.
- If 0, `span` equals `other`.
- If greater than 0, `span` follows `other`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

SequenceCompareTo<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Determines the relative order of two read-only sequences by comparing their elements using IComparable{T}.CompareTo(T).

C#

```
public static int SequenceCompareTo<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other) where T : IComparable<T>;
```

Type Parameters

`T`

The type of elements in the sequence.

Parameters

`span` ReadOnlySpan<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

Returns

Int32

A signed integer that indicates the relative order of `span` and `other`:
- If less than 0, `span` precedes than `other`.
- If 0, `span` equals `other`.
- If greater than 0, `span` follows `other`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.SequenceEqual Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

SequenceEqual<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
SequenceEqual<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span are equal by comparing the elements using IEquatable{T}.Equals(T).
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences are equal by comparing the elements using IEquatable{T}.Equals(T).

SequenceEqual<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)

Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.

C#

```
public static bool SequenceEqual<T> (this Span<T> span, ReadOnlySpan<T>
other, System.Collections.Generic.IEqualityComparer<T>? comparer =
default);
```

Type Parameters

`T`

The type of elements in the sequence.

Parameters

`span` Span<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing elements, or `null` to use the default IEqualityComparer<T> for the type of an element.

Returns

Boolean

`true` if the two sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)

Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.

C#

```
public static bool SequenceEqual<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other, System.Collections.Generic.IEqualityComparer<T>?
```

```
comparer = default);
```

Type Parameters

`T`

The type of elements in the sequence.

Parameters

`span` ReadOnlySpan<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing elements, or `null` to use the default IEqualityComparer<T> for the type of an element.

Returns

Boolean

`true` if the two sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

SequenceEqual<T>(Span<T>, ReadOnlySpan<T>)

Determines whether a span and a read-only span are equal by comparing the elements using IEquatable{T}.Equals(T).

```
C#
```

```
public static bool SequenceEqual<T> (this Span<T> span, ReadOnlySpan<T>
other) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of elements in the sequence.

Parameters

`span` Span<T>

The span to compare.

`other` ReadOnlySpan<T>

The read-only span to compare.

Returns

Boolean

`true` if the two sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Determines whether two read-only sequences are equal by comparing the elements using IEquatable{T}.Equals(T).

```
C#
```

```
public static bool SequenceEqual<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> other) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of elements in the sequence.

Parameters

`span` ReadOnlySpan<T>

The first sequence to compare.

`other` ReadOnlySpan<T>

The second sequence to compare.

Returns

Boolean

`true` if the two sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

MemoryExtensions.Sort Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Sort<T,TComparer>(Span<T>, TComparer)	Sorts the elements in the entire Span<T> using the `TComparer`.
Sort<T>(Span<T>)	Sorts the elements in the entire Span<T> using the IComparable<T> implementation of each element of the Span<T>.
Sort<T>(Span<T>, Comparison<T>)	Sorts the elements in the entire Span<T> using the specified Comparison<T>.
Sort<TKey,TValue,TComparer>(Span<TKey>, Span<TValue>, TComparer)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparer.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the IComparable<T> implementation of each key.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>, Comparison<TKey>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparison.

Sort<T,TComparer>(Span<T>, TComparer)

Sorts the elements in the entire Span<T> using the `TComparer`.

```
C#
```

```
public static void Sort<T,TComparer> (this Span<T> span, TComparer
comparer) where TComparer : System.Collections.Generic.IComparer<T>;
```

Type Parameters

`T`

The type of the elements of the span.

`TComparer`

The type of the comparer to use to compare elements.

Parameters

`span` Span<T>

The span of memory to sort.

`comparer` TComparer

The method to use when comparing elements, or `null` to use each element's IComparable<T> interface implementation.

Exceptions

InvalidOperationException

`comparer` is `null`, and one or more elements in `span` do not implement the IComparable<T> interface.

ArgumentException

The implementation of `comparer` caused an error during the sort.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Sort<T>(Span<T>)

Sorts the elements in the entire Span<T> using the IComparable<T> implementation of each element of the Span<T>.

```
C#
public static void Sort<T> (this Span<T> span);
```

Type Parameters

T

The type of the elements of the span.

Parameters

span Span<T>

The span of memory to sort.

Exceptions

InvalidOperationException

One or more elements in `span` do not implement the IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Sort<T>(Span<T>, Comparison<T>)

Sorts the elements in the entire Span<T> using the specified Comparison<T>.

```
C#
public static void Sort<T> (this Span<T> span, Comparison<T> comparison);
```

Type Parameters

T

The type of the elements of the span.

Parameters

`span` Span<T>

The span of memory to sort.

`comparison` Comparison<T>

The method to use when comparing elements.

Exceptions

ArgumentNullException

`comparison` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Sort<TKey,TValue,TComparer>(Span<TKey>, Span<TValue>, TComparer)

Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparer.

C#

```
public static void Sort<TKey,TValue,TComparer> (this Span<TKey> keys,
Span<TValue> items, TComparer comparer) where TComparer :
System.Collections.Generic.IComparer<TKey>;
```

Type Parameters

`TKey`

The type of the elements of the key span.

`TValue`

The type of the elements of the items span.

`TComparer`

The type of the comparer to use to compare elements.

Parameters

`keys` Span<TKey>

The span that contains the keys to sort.

`items` Span<TValue>

The span that contains the items that correspond to the keys in `keys`.

`comparer` TComparer

The IComparer<T> implementation to use when comparing elements, or `null` to use the IComparable<T> interface implementation of each element.

Exceptions

ArgumentException

The length of `keys` isn't equal to the length of `items`.

InvalidOperationException

`comparer` is `null`, and one or more elements in `keys` do not implement the IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Sort<TKey,TValue>(Span<TKey>, Span<TValue>)

Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the IComparable<T> implementation of each key.

```
C#

public static void Sort<TKey,TValue> (this Span<TKey> keys, Span<TValue>
items);
```

Type Parameters

`TKey`

The type of the elements of the key span.

`TValue`

The type of the elements of the items span.

Parameters

`keys` Span<TKey>

The span that contains the keys to sort.

`items` Span<TValue>

The span that contains the items that correspond to the keys in `keys`.

Exceptions

ArgumentException

The length of `keys` isn't equal to the length of `items`.

InvalidOperationException

One or more elements in `keys` do not implement the IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Sort<TKey,TValue>(Span<TKey>, Span<TValue>, Comparison<TKey>)

Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparison.

```
C#

public static void Sort<TKey,TValue> (this Span<TKey> keys, Span<TValue>
items, Comparison<TKey> comparison);
```

Type Parameters

`TKey`

The type of the elements of the key span.

`TValue`

The type of the elements of the items span.

Parameters

`keys` Span<TKey>

The span that contains the keys to sort.

`items` Span<TValue>

The span that contains the items that correspond to the keys in `keys`.

`comparison` Comparison<TKey>

The Comparison<T> to use when comparing elements.

Exceptions

ArgumentNullException

`comparison` is `null`.

ArgumentException

The length of `keys` isn't equal to the length of `items`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Split Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Split(ReadOnlySpan<Char>, Span<Range>, Char, StringSplitOptions)	Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.
Split(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.

Split(ReadOnlySpan<Char>, Span<Range>, Char, StringSplitOptions)

Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.

C#

```
public static int Split (this ReadOnlySpan<char> source, Span<Range>
destination, char separator, StringSplitOptions options =
System.StringSplitOptions.None);
```

Parameters

source ReadOnlySpan<Char>

The source span to parse.

`destination` Span<Range>

The destination span into which the resulting ranges are written.

`separator` Char

A character that delimits the regions in this instance.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim whitespace and include empty ranges.

Returns

Int32

The number of ranges written into `destination`.

Remarks

Delimiter characters are not included in the elements of the returned array.

If the `destination` span is empty, or if the `options` specifies RemoveEmptyEntries and `source` is empty, or if `options` specifies both RemoveEmptyEntries and TrimEntries and the `source` is entirely whitespace, no ranges are written to the destination.

If the span does not contain `separator`, or if `destination`'s length is 1, a single range will be output containing the entire `source`, subject to the processing implied by `options`.

If there are more regions in `source` than will fit in `destination`, the first `destination` length minus 1 ranges are stored in `destination`, and a range for the remainder of `source` is stored in `destination`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Split(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)

Parses the source ReadOnlySpan<T> for the specified `separator`, populating the `destination` span with Range instances representing the regions between the separators.

```
C#

public static int Split (this ReadOnlySpan<char> source, Span<Range>
destination, ReadOnlySpan<char> separator, StringSplitOptions options =
System.StringSplitOptions.None);
```

Parameters

`source` ReadOnlySpan<Char>

The source span to parse.

`destination` Span<Range>

The destination span into which the resulting ranges are written.

`separator` ReadOnlySpan<Char>

A character that delimits the regions in this instance.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim whitespace and include empty ranges.

Returns

Int32

The number of ranges written into `destination`.

Remarks

Delimiter characters are not included in the elements of the returned array.

If the `destination` span is empty, or if the `options` specifies RemoveEmptyEntries and `source` is empty, or if `options` specifies both RemoveEmptyEntries and

TrimEntries and the `source` is entirely whitespace, no ranges are written to the destination.

If the span does not contain `separator`, or if `destination`'s length is 1, a single range will be output containing the entire `source`, subject to the processing implied by `options`.

If there are more regions in `source` than will fit in `destination`, the first `destination` length minus 1 ranges are stored in `destination`, and a range for the remainder of `source` is stored in `destination`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.SplitAny Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<String>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.
SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)	Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.

SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<String>, StringSplitOptions)

Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.

C#

```
public static int SplitAny (this ReadOnlySpan<char> source, Span<Range>
destination, ReadOnlySpan<string> separators, StringSplitOptions options
= System.StringSplitOptions.None);
```

Parameters

`source` ReadOnlySpan<Char>

The source span to parse.

`destination` Span<Range>

The destination span into which the resulting ranges are written.

`separators` ReadOnlySpan<String>

Any number of strings that may delimit the regions in this instance. If empty, all Unicode whitespace characters are used as the separators.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim whitespace and include empty ranges.

Returns

Int32

The number of ranges written into `destination`.

Remarks

Delimiter characters are not included in the elements of the returned array.

If the `destination` span is empty, or if the `options` specifies RemoveEmptyEntries and `source` is empty, or if `options` specifies both RemoveEmptyEntries and TrimEntries and the `source` is entirely whitespace, no ranges are written to the destination.

If the span does not contain any of the `separators`, or if `destination`'s length is 1, a single range will be output containing the entire `source`, subject to the processing implied by `options`.

If there are more regions in `source` than will fit in `destination`, the first `destination` length minus 1 ranges are stored in `destination`, and a range for the remainder of `source` is stored in `destination`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

SplitAny(ReadOnlySpan<Char>, Span<Range>, ReadOnlySpan<Char>, StringSplitOptions)

Parses the source ReadOnlySpan<T> for one of the specified `separators`, populating the `destination` span with Range instances representing the regions between the separators.

```
C#

public static int SplitAny (this ReadOnlySpan<char> source, Span<Range>
destination, ReadOnlySpan<char> separators, StringSplitOptions options =
System.StringSplitOptions.None);
```

Parameters

`source` ReadOnlySpan<Char>

The source span to parse.

`destination` Span<Range>

The destination span into which the resulting ranges are written.

`separators` ReadOnlySpan<Char>

Any number of characters that may delimit the regions in this instance. If empty, all Unicode whitespace characters are used as the separators.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim whitespace and include empty ranges.

Returns

Int32

The number of ranges written into `destination`.

Remarks

Delimiter characters are not included in the elements of the returned array.

If the `destination` span is empty, or if the `options` specifies RemoveEmptyEntries and `source` is empty, or if `options` specifies both RemoveEmptyEntries and

TrimEntries and the `source` is entirely whitespace, no ranges are written to the destination.

If the span does not contain any of the `separators`, or if `destination`'s length is 1, a single range will be output containing the entire `source`, subject to the processing implied by `options`.

If there are more regions in `source` than will fit in `destination`, the first `destination` length minus 1 ranges are stored in `destination`, and a range for the remainder of `source` is stored in `destination`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.StartsWith Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

StartsWith(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)	Determines whether a read-only character span begins with a specified value when compared using a specified StringComparison value.
StartsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a specified sequence appears at the start of a span.
StartsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether a specified sequence appears at the start of a read-only span.

StartsWith(ReadOnlySpan<Char>, ReadOnlySpan<Char>, StringComparison)

Determines whether a read-only character span begins with a specified value when compared using a specified StringComparison value.

C#

```
public static bool StartsWith (this ReadOnlySpan<char> span,
ReadOnlySpan<char> value, StringComparison comparisonType);
```

Parameters

span ReadOnlySpan<Char>

The source span.

value ReadOnlySpan<Char>

The sequence to compare to the beginning of the source span.

`comparisonType` StringComparison

An enumeration value that determines how `span` and `value` are compared.

Returns

Boolean

`true` if `value` matches the beginning of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

StartsWith<T>(Span<T>, ReadOnlySpan<T>)

Determines whether a specified sequence appears at the start of a span.

C#

```
public static bool StartsWith<T> (this Span<T> span, ReadOnlySpan<T>
value) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The span to search.

`value` ReadOnlySpan<T>

A sequence to search for at the start of `span`.

Returns

Boolean

`true` if `value` matches the beginning of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

StartsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Determines whether a specified sequence appears at the start of a read-only span.

C#

```
public static bool StartsWith<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> value) where T : IEquatable<T>;
```

Type Parameters

T

The type of elements in the span.

Parameters

span ReadOnlySpan<T>

The read-only character span to search.

value ReadOnlySpan<T>

A sequence to search for at the start of `span`.

Returns

Boolean

`true` if `value` matches the beginning of `span`; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ToLower Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Copies the characters from the source span into the destination, converting each character to lowercase, using the casing rules of the specified culture.

C#

```
public static int ToLower (this ReadOnlySpan<char> source, Span<char>
destination, System.Globalization.CultureInfo? culture);
```

Parameters

source ReadOnlySpan<Char>

The source span.

destination Span<Char>

The destination span which contains the transformed characters.

culture CultureInfo

An object that supplies culture-specific casing rules.

Returns

Int32

The number of characters written into the destination span. If the destination is too small, returns -1.

Exceptions

InvalidOperationException

The source and destination buffers overlap.

Remarks

If `culture` is `null`, CultureInfo.CurrentCulture will be used.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ToLowerInvariant Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Copies the characters from the source span into the destination, converting each character to lowercase, using the casing rules of the invariant culture.

C#

```
public static int ToLowerInvariant (this ReadOnlySpan<char> source,
Span<char> destination);
```

Parameters

source ReadOnlySpan<Char>

The source span.

destination Span<Char>

The destination span which contains the transformed characters.

Returns

Int32

The number of characters written into the destination span. If the destination is too small, returns -1.

Exceptions

InvalidOperationException

The source and destination buffers overlap.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ToUpper Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Copies the characters from the source span into the destination, converting each character to uppercase, using the casing rules of the specified culture.

C#

```
public static int ToUpper (this ReadOnlySpan<char> source, Span<char>
destination, System.Globalization.CultureInfo? culture);
```

Parameters

source ReadOnlySpan<Char>

The source span.

destination Span<Char>

The destination span which contains the transformed characters.

culture CultureInfo

An object that supplies culture-specific casing rules.

Returns

Int32

The number of characters written into the destination span. If the destination is too small, returns -1.

Exceptions

InvalidOperationException

The source and destination buffers overlap.

Remarks

If `culture` is `null`, CultureInfo.CurrentCulture will be used.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.ToUpperInvariant Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Copies the characters from the source span into the destination, converting each character to uppercase using the casing rules of the invariant culture.

C#

```
public static int ToUpperInvariant (this ReadOnlySpan<char> source,
Span<char> destination);
```

Parameters

`source` ReadOnlySpan<Char>

The source span.

`destination` Span<Char>

The destination span which contains the transformed characters.

Returns

Int32

The number of characters written into the destination span. If the destination is too small, returns -1.

Exceptions

InvalidOperationException

The source and destination buffers overlap.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.Trim Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Trim(Memory<Char>)	Removes all leading and trailing whitespace characters from a character memory region.
Trim(ReadOnlyMemory<Char>)	Removes all leading and trailing whitespace characters from a read-only character memory region.
Trim(ReadOnlySpan<Char>)	Removes all leading and trailing whitespace characters from a read-only character span.
Trim(Span<Char>)	Removes all leading and trailing whitespace characters from a character span.
Trim(ReadOnlySpan<Char>, Char)	Removes all leading and trailing occurrences of a specified character from a read-only character span.
Trim(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Removes all leading and trailing occurrences of a set of characters specified in a read-only span from a read-only character span.
Trim<T>(ReadOnlyMemory<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only memory region.
Trim<T>(Span<T>, T)	Removes all leading and trailing occurrences of a specified element from a span.
Trim<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a span.
Trim<T>(ReadOnlySpan<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only span.
Trim<T>(Memory<T>, T)	Removes all leading and trailing occurrences of a specified element from a memory region.

Trim<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
Trim<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a memory region.
Trim<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only span.

Trim(Memory<Char>)

Removes all leading and trailing whitespace characters from a character memory region.

C#

```
public static Memory<char> Trim (this Memory<char> memory);
```

Parameters

`memory` Memory<Char>

The source memory from which the characters are removed.

Returns

Memory<Char>

The trimmed character memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim(ReadOnlyMemory<Char>)

Removes all leading and trailing whitespace characters from a read-only character memory region.

```
C#

public static ReadOnlyMemory<char> Trim (this ReadOnlyMemory<char>
memory);
```

Parameters

`memory` ReadOnlyMemory<Char>

The source memory from which the characters are removed.

Returns

ReadOnlyMemory<Char>

The trimmed character memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim(ReadOnlySpan<Char>)

Removes all leading and trailing whitespace characters from a read-only character span.

```
C#

public static ReadOnlySpan<char> Trim (this ReadOnlySpan<char> span);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the characters are removed.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Trim(Span<Char>)

Removes all leading and trailing whitespace characters from a character span.

C#

```
public static Span<char> Trim (this Span<char> span);
```

Parameters

span Span<Char>

The source span from which the characters are removed.

Returns

Span<Char>

The trimmed character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim(ReadOnlySpan<Char>, Char)

Removes all leading and trailing occurrences of a specified character from a read-only character span.

```
C#

public static ReadOnlySpan<char> Trim (this ReadOnlySpan<char> span, char
trimChar);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the character is removed.

`trimChar` Char

The specified character to look for and remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Trim(ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Removes all leading and trailing occurrences of a set of characters specified in a read-only span from a read-only character span.

```
C#
```

```
public static ReadOnlySpan<char> Trim (this ReadOnlySpan<char> span,
ReadOnlySpan<char> trimChars);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the characters are removed.

`trimChars` ReadOnlySpan<Char>

The span which contains the set of characters to remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Remarks

If `trimChars` is empty, whitespace characters are removed instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Trim<T>(ReadOnlyMemory<T>, T)

Removes all leading and trailing occurrences of a specified element from a read-only memory region.

C#

```
public static ReadOnlyMemory<T> Trim<T> (this ReadOnlyMemory<T> memory, T
trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only memory region.

Parameters

`memory` ReadOnlyMemory<T>

The source memory from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(Span<T>, T)

Removes all leading and trailing occurrences of a specified element from a span.

```
C#

public static Span<T> Trim<T> (this Span<T> span, T trimElement) where T
: IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The source span from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

Span<T>

The trimmed span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(Span<T>, ReadOnlySpan<T>)

Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a span.

C#

```
public static Span<T> Trim<T> (this Span<T> span, ReadOnlySpan<T>
trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The source span from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Span<T>

The trimmed span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(ReadOnlySpan<T>, T)

Removes all leading and trailing occurrences of a specified element from a read-only span.

C#

```
public static ReadOnlySpan<T> Trim<T> (this ReadOnlySpan<T> span, T
trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only span.

Parameters

`span` ReadOnlySpan<T>

The source span from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(Memory<T>, T)

Removes all leading and trailing occurrences of a specified element from a memory region.

C#

```
public static Memory<T> Trim<T> (this Memory<T> memory, T trimElement)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the memory region.

Parameters

`memory` Memory<T>

The source memory from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

Memory<T>

The trimmed memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)

Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.

C#

```
public static ReadOnlyMemory<T> Trim<T> (this ReadOnlyMemory<T> memory,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only memory region.

Parameters

`memory` ReadOnlyMemory<T>

The source memory from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(Memory<T>, ReadOnlySpan<T>)

Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a memory region.

```
C#

public static Memory<T> Trim<T> (this Memory<T> memory, ReadOnlySpan<T>
trimElements) where T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the memory region.

Parameters

memory Memory<T>

The source memory from which the elements are removed.

trimElements ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Memory<T>

The trimmed memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Trim<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only span.

C#

```
public static ReadOnlySpan<T> Trim<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the read-only span.

Parameters

span ReadOnlySpan<T>

The source span from which the elements are removed.

trimElements ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TrimEnd Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

TrimEnd(ReadOnlySpan<Char>, Char)	Removes all trailing occurrences of a specified character from a read-only span.
TrimEnd(Memory<Char>)	Removes all trailing whitespace characters from a character memory region.
TrimEnd(ReadOnlyMemory<Char>)	Removes all trailing whitespace characters from a read-only character memory region.
TrimEnd(ReadOnlySpan<Char>)	Removes all trailing whitespace characters from a read-only character span.
TrimEnd(Span<Char>)	Removes all trailing whitespace characters from a character span.
TrimEnd(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Removes all trailing occurrences of a set of characters specified in a read-only span from a read-only character span.
TrimEnd<T>(Memory<T>, T)	Removes all trailing occurrences of a specified element from a character memory region.
TrimEnd<T>(Span<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a span.
TrimEnd<T>(Span<T>, T)	Removes all trailing occurrences of a specified element from a span.
TrimEnd<T>(ReadOnlySpan<T>, T)	Removes all trailing occurrences of a specified element from a read-only span.
TrimEnd<T>(Memory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a memory region.

TrimEnd<T>(ReadOnlyMemory<T>, T)	Removes all trailing occurrences of a specified element from a read-only memory region.
TrimEnd<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
TrimEnd<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only span.

TrimEnd(ReadOnlySpan<Char>, Char)

Removes all trailing occurrences of a specified character from a read-only span.

C#

```
public static ReadOnlySpan<char> TrimEnd (this ReadOnlySpan<char> span,
char trimChar);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the character is removed.

`trimChar` Char

The specified character to look for and remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimEnd(Memory<Char>)

Removes all trailing whitespace characters from a character memory region.

C#

```
public static Memory<char> TrimEnd (this Memory<char> memory);
```

Parameters

memory Memory<Char>

The source memory from which the characters are removed.

Returns

Memory<Char>

The trimmed character memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd(ReadOnlyMemory<Char>)

Removes all trailing whitespace characters from a read-only character memory region.

C#

```
public static ReadOnlyMemory<char> TrimEnd (this ReadOnlyMemory<char>
memory);
```

Parameters

memory ReadOnlyMemory<Char>

The source memory from which the characters are removed.

Returns

ReadOnlyMemory<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd(ReadOnlySpan<Char>)

Removes all trailing whitespace characters from a read-only character span.

C#

```
public static ReadOnlySpan<char> TrimEnd (this ReadOnlySpan<char> span);
```

Parameters

span ReadOnlySpan<Char>

The source span from which the characters are removed.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimEnd(Span<Char>)

Removes all trailing whitespace characters from a character span.

C#

```
public static Span<char> TrimEnd (this Span<char> span);
```

Parameters

span Span<Char>

The source span from which the characters are removed.

Returns

Span<Char>

The trimmed character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd(ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Removes all trailing occurrences of a set of characters specified in a read-only span from a read-only character span.

C#

```
public static ReadOnlySpan<char> TrimEnd (this ReadOnlySpan<char> span,
ReadOnlySpan<char> trimChars);
```

Parameters

span ReadOnlySpan<Char>

The source span from which the characters are removed.

`trimChars` ReadOnlySpan<Char>

The span which contains the set of characters to remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Remarks

If `trimChars` is empty, whitespace characters are removed instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimEnd<T>(Memory<T>, T)

Removes all trailing occurrences of a specified element from a character memory region.

```
C#

public static Memory<T> TrimEnd<T> (this Memory<T> memory, T trimElement)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the memory region.

Parameters

`memory` Memory<T>

The source memory from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

Memory<T>

The trimmed memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(Span<T>, ReadOnlySpan<T>)

Removes all trailing occurrences of a set of elements specified in a read-only span from a span.

C#

```
public static Span<T> TrimEnd<T> (this Span<T> span, ReadOnlySpan<T>
trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The source span from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Span<T>

The trimmed span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(Span<T>, T)

Removes all trailing occurrences of a specified element from a span.

C#

```
public static Span<T> TrimEnd<T> (this Span<T> span, T trimElement) where
T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the span.

Parameters

span Span<T>

The source span from which the element is removed.

trimElement T

The specified element to look for and remove.

Returns

Span<T>

The trimmed span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(ReadOnlySpan<T>, T)

Removes all trailing occurrences of a specified element from a read-only span.

C#

```
public static ReadOnlySpan<T> TrimEnd<T> (this ReadOnlySpan<T> span, T
trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only span.

Parameters

`span` ReadOnlySpan<T>

The source span from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(Memory<T>, ReadOnlySpan<T>)

Removes all trailing occurrences of a set of elements specified in a read-only span from a memory region.

C#

```
public static Memory<T> TrimEnd<T> (this Memory<T> memory,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the memory region.

Parameters

`memory` Memory<T>

The source memory from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Memory<T>

The trimmed memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(ReadOnlyMemory<T>, T)

Removes all trailing occurrences of a specified element from a read-only memory region.

C#

```
public static ReadOnlyMemory<T> TrimEnd<T> (this ReadOnlyMemory<T>
memory, T trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only memory region.

Parameters

`memory` ReadOnlyMemory<T>

The source memory from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)

Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.

C#

```
public static ReadOnlyMemory<T> TrimEnd<T> (this ReadOnlyMemory<T>
memory, ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only memory region.

Parameters

`memory` ReadOnlyMemory<T>

The source memory from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimEnd<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only span.

C#

```
public static ReadOnlySpan<T> TrimEnd<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only span.

Parameters

`span` ReadOnlySpan<T>

The source span from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TrimStart Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

TrimStart(Memory<Char>)	Removes all leading whitespace characters from a memory region.
TrimStart(ReadOnlyMemory<Char>)	Removes all leading whitespace characters from a read-only memory region.
TrimStart(ReadOnlySpan<Char>)	Removes all leading whitespace characters from a read-only span.
TrimStart(Span<Char>)	Removes all leading whitespace characters from a span.
TrimStart(ReadOnlySpan<Char>, Char)	Removes all leading occurrences of a specified character from the span.
TrimStart(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Removes all leading occurrences of a set of characters specified in a read-only span from the span.
TrimStart<T>(Span<T>, T)	Removes all leading occurrences of a specified element from the span.
TrimStart<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from the span.
TrimStart<T>(ReadOnlySpan<T>, T)	Removes all leading occurrences of a specified element from the span.
TrimStart<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from the span.

TrimStart<T>(Memory<T>, T)	Removes all leading occurrences of a specified element from a memory region.
TrimStart<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.
TrimStart<T>(Memory<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.
TrimStart<T>(ReadOnlyMemory<T>, T)	Removes all leading occurrences of a specified element from a memory region.

TrimStart(Memory<Char>)

Removes all leading whitespace characters from a memory region.

```
C#

public static Memory<char> TrimStart (this Memory<char> memory);
```

Parameters

`memory` Memory<Char>

The source memory from which the characters are removed.

Returns

Memory<Char>

The trimmed character memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart(ReadOnlyMemory<Char>)

Removes all leading whitespace characters from a read-only memory region.

```
C#

public static ReadOnlyMemory<char> TrimStart (this ReadOnlyMemory<char>
memory);
```

Parameters

memory ReadOnlyMemory<Char>

The source memory from which the characters are removed.

Returns

ReadOnlyMemory<Char>

The trimmed read-only character memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart(ReadOnlySpan<Char>)

Removes all leading whitespace characters from a read-only span.

```
C#

public static ReadOnlySpan<char> TrimStart (this ReadOnlySpan<char>
span);
```

Parameters

span ReadOnlySpan<Char>

The source span from which the characters are removed.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimStart(Span<Char>)

Removes all leading whitespace characters from a span.

C#

```
public static Span<char> TrimStart (this Span<char> span);
```

Parameters

span Span<Char>

The source span from which the characters are removed.

Returns

Span<Char>

The trimmed character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart(ReadOnlySpan<Char>, Char)

Removes all leading occurrences of a specified character from the span.

C#

```
public static ReadOnlySpan<char> TrimStart (this ReadOnlySpan<char> span,
char trimChar);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the character is removed.

`trimChar` Char

The specified character to look for and remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimStart(ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Removes all leading occurrences of a set of characters specified in a read-only span from the span.

C#

```
public static ReadOnlySpan<char> TrimStart (this ReadOnlySpan<char> span,
ReadOnlySpan<char> trimChars);
```

Parameters

`span` ReadOnlySpan<Char>

The source span from which the characters are removed.

`trimChars` ReadOnlySpan<Char>

The span which contains the set of characters to remove.

Returns

ReadOnlySpan<Char>

The trimmed read-only character span.

Remarks

If `trimChars` is empty, whitespace characters are removed instead.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimStart<T>(Span<T>, T)

Removes all leading occurrences of a specified element from the span.

C#

```
public static Span<T> TrimStart<T> (this Span<T> span, T trimElement)
where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The source span from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

Span<T>

The trimmed span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(Span<T>, ReadOnlySpan<T>)

Removes all leading occurrences of a set of elements specified in a read-only span from the span.

C#

```
public static Span<T> TrimStart<T> (this Span<T> span, ReadOnlySpan<T>
trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the span.

Parameters

`span` Span<T>

The source span from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Span<T>

The trimmed span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(ReadOnlySpan<T>, T)

Removes all leading occurrences of a specified element from the span.

```
C#

public static ReadOnlySpan<T> TrimStart<T> (this ReadOnlySpan<T> span, T
trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only span.

Parameters

`span` ReadOnlySpan<T>

The source span from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)

Removes all leading occurrences of a set of elements specified in a read-only span from the span.

C#

```
public static ReadOnlySpan<T> TrimStart<T> (this ReadOnlySpan<T> span,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only span.

Parameters

`span` ReadOnlySpan<T>

The source span from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlySpan<T>

The trimmed read-only span.

Remarks

If `trimElements` is empty, the span is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(Memory<T>, T)

Removes all leading occurrences of a specified element from a memory region.

```
C#

public static Memory<T> TrimStart<T> (this Memory<T> memory, T
trimElement) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the memory region.

Parameters

`memory` Memory<T>

The source memory region from which the element is removed.

`trimElement` T

The specified element to look for and remove.

Returns

Memory<T>

The trimmed memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)

Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.

C#

```
public static ReadOnlyMemory<T> TrimStart<T> (this ReadOnlyMemory<T>
memory, ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

`T`

The type of the elements in the read-only memory region.

Parameters

`memory` ReadOnlyMemory<T>

The source memory from which the elements are removed.

`trimElements` ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(Memory<T>, ReadOnlySpan<T>)

Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.

C#

```
public static Memory<T> TrimStart<T> (this Memory<T> memory,
ReadOnlySpan<T> trimElements) where T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the memory region.

Parameters

memory Memory<T>

The source memory from which the elements are removed.

trimElements ReadOnlySpan<T>

The span which contains the set of elements to remove.

Returns

Memory<T>

The trimmed memory region.

Remarks

If `trimElements` is empty, the memory is returned unaltered.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

TrimStart<T>(ReadOnlyMemory<T>, T)

Removes all leading occurrences of a specified element from a memory region.

C#

```
public static ReadOnlyMemory<T> TrimStart<T> (this ReadOnlyMemory<T>
memory, T trimElement) where T : IEquatable<T>;
```

Type Parameters

T

The type of the elements in the read-only memory region.

Parameters

memory ReadOnlyMemory<T>

The source memory from which the element is removed.

trimElement T

The specified element to look for and remove.

Returns

ReadOnlyMemory<T>

The trimmed read-only memory region.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TryWrite Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, Object[])	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite(Span<Char>, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)	Writes the specified interpolated string to the character span.
TryWrite(Span<Char>, IFormatProvider, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)	Writes the specified interpolated string to the character span.
TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, ReadOnlySpan<Object>)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite<TArg0,TArg1,TArg2>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1, TArg2)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.
TryWrite<TArg0,TArg1>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string

	representation of the corresponding arguments.
TryWrite<TArg0>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0)	Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, Object[])

Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

C#

```
public static bool TryWrite (this Span<char> destination,
IFormatProvider? provider, System.Text.CompositeFormat format, out int
charsWritten, params object?[] args);
```

Parameters

destination Span<Char>

The span to which the string should be formatted.

provider IFormatProvider

An object that supplies culture-specific formatting information.

format CompositeFormat

A CompositeFormat.

charsWritten Int32

When this method returns, contains the number of characters written to the span.

args Object[]

An array of objects to format.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Exceptions

ArgumentNullException

`format` or `args` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryWrite(Span<Char>, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)

Writes the specified interpolated string to the character span.

C#

```
public static bool TryWrite (this Span<char> destination, ref
MemoryExtensions.TryWriteInterpolatedStringHandler handler, out int
charsWritten);
```

Parameters

`destination` Span<Char>

The span to which the interpolated string should be formatted.

`handler` MemoryExtensions.TryWriteInterpolatedStringHandler

The interpolated string, passed by reference.

`charsWritten` Int32

When this method returns, contains the number of characters written to the span.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryWrite(Span<Char>, IFormatProvider, MemoryExtensions+TryWriteInterpolatedStringHandler, Int32)

Writes the specified interpolated string to the character span.

C#

```
public static bool TryWrite (this Span<char> destination,
IFormatProvider? provider, ref
MemoryExtensions.TryWriteInterpolatedStringHandler handler, out int
charsWritten);
```

Parameters

`destination` Span<Char>

The span to which the interpolated string should be formatted.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`handler` MemoryExtensions.TryWriteInterpolatedStringHandler

The interpolated string, passed by reference.

`charsWritten` Int32

When this method returns, contains the number of characters written to the span.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryWrite(Span<Char>, IFormatProvider, CompositeFormat, Int32, ReadOnlySpan<Object>)

Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

C#

```
public static bool TryWrite (this Span<char> destination,
IFormatProvider? provider, System.Text.CompositeFormat format, out int
charsWritten, ReadOnlySpan<object?> args);
```

Parameters

`destination` Span<Char>

The span to which the string should be formatted.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`charsWritten` Int32

When this method returns, contains the number of characters written to the span.

`args` ReadOnlySpan<Object>

A span of objects to format.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryWrite<TArg0,TArg1,TArg2>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1, TArg2)

Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

C#

```
public static bool TryWrite<TArg0,TArg1,TArg2> (this Span<char>
destination, IFormatProvider? provider, System.Text.CompositeFormat
format, out int charsWritten, TArg0 arg0, TArg1 arg1, TArg2 arg2);
```

Type Parameters

TArg0

The type of the first object to format.

TArg1

The type of the second object to format.

TArg2

The type of the third object to format.

Parameters

destination Span<Char>

The span to which the string should be formatted.

provider IFormatProvider

An object that supplies culture-specific formatting information.

format CompositeFormat

A CompositeFormat.

charsWritten Int32

When this method returns, contains the number of characters written to the span.

arg0 TArg0

The first object to format.

arg1 TArg1

The second object to format.

arg2 TArg2

The third object to format.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryWrite<TArg0,TArg1>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0, TArg1)

Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

C#

```
public static bool TryWrite<TArg0,TArg1> (this Span<char> destination,
IFormatProvider? provider, System.Text.CompositeFormat format, out int
charsWritten, TArg0 arg0, TArg1 arg1);
```

Type Parameters

`TArg0`

The type of the first object to format.

`TArg1`

The type of the second object to format.

Parameters

`destination` Span<Char>

The span to which the string should be formatted.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`charsWritten` Int32

When this method returns, contains the number of characters written to the span.

`arg0` TArg0

The first object to format.

`arg1` TArg1

The second object to format.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryWrite<TArg0>(Span<Char>, IFormatProvider, CompositeFormat, Int32, TArg0)

Writes the CompositeFormat string to the character span, substituting the format item or items with the string representation of the corresponding arguments.

C#

```
public static bool TryWrite<TArg0> (this Span<char> destination,
IFormatProvider? provider, System.Text.CompositeFormat format, out int
charsWritten, TArg0 arg0);
```

Type Parameters

`TArg0`

The type of the object to format.

Parameters

`destination` Span<Char>

The span to which the string should be formatted.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`charsWritten` Int32

When this method returns, contains the number of characters written to the span.

`arg0` TArg0

The object to format.

Returns

Boolean

`true` if the entire interpolated string could be formatted successfully; otherwise, `false`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TryWriteInterpolatedStringHandler Struct

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Provides a handler used by the language compiler to format interpolated strings into character spans.

C#

```
public ref struct MemoryExtensions.TryWriteInterpolatedStringHandler
```

Inheritance Object → ValueType → MemoryExtensions.TryWriteInterpolatedStringHandler

Constructors

Expand table

MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, Boolean)	Creates a handler used to write an interpolated string into a span of characters.
MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, IFormatProvider, Boolean)	Creates a handler used to write an interpolated string into a span of characters.

Methods

Expand table

AppendFormatted(Object, Int32, String)	Writes the specified value to the handler.
AppendFormatted(ReadOnlySpan<Char>)	Writes the specified value to the handler.

AppendFormatted(ReadOnlySpan<Char>, Int32, String)	Writes the specified value to the handler.
AppendFormatted(String)	Writes the specified value to the handler.
AppendFormatted(String, Int32, String)	Writes a specified value to the handler using a specified format string.
AppendFormatted<T>(T)	Writes the specified value to the handler.
AppendFormatted<T>(T, Int32)	Writes the specified value to the handler.
AppendFormatted<T>(T, Int32, String)	Writes a specified value to the handler using a specified format string.
AppendFormatted<T>(T, String)	Writes a specified value to the handler using a specified format string.
AppendLiteral(String)	Writes the specified string to the handler.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TryWriteInterpolatedStringHandler Constructors

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, Boolean)	Creates a handler used to write an interpolated string into a span of characters.
MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, IFormatProvider, Boolean)	Creates a handler used to write an interpolated string into a span of characters.

MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, Boolean)

Creates a handler used to write an interpolated string into a span of characters.

C#

```
public TryWriteInterpolatedStringHandler (int literalLength, int
formattedCount, Span<char> destination, out bool shouldAppend);
```

Parameters

`literalLength` Int32

The number of constant characters outside of interpolation expressions in the interpolated string.

`formattedCount` Int32

The number of interpolation expressions in the interpolated string.

`destination` Span<Char>

The destination buffer.

`shouldAppend` Boolean

When this method returns, contains `true` if the destination may be long enough to support the formatting, or `false` if it won't be.

Remarks

This is intended to be called only by compiler-generated code. Arguments are not validated as they'd otherwise be for members intended to be used directly.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

MemoryExtensions.TryWriteInterpolatedStringHandler(Int32, Int32, Span<Char>, IFormatProvider, Boolean)

Creates a handler used to write an interpolated string into a span of characters.

C#

```
public TryWriteInterpolatedStringHandler (int literalLength, int
formattedCount, Span<char> destination, IFormatProvider? provider, out
bool shouldAppend);
```

Parameters

`literalLength` Int32

The number of constant characters outside of interpolation expressions in the interpolated string.

`formattedCount` Int32

The number of interpolation expressions in the interpolated string.

`destination` Span<Char>

The destination buffer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`shouldAppend` Boolean

When this method returns, contains `true` if the destination may be long enough to support the formatting, or `false` if it won't be.

Remarks

This is intended to be called only by compiler-generated code. Arguments are not validated as they'd otherwise be for members intended to be used directly.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TryWriteInterpolatedStringHandler.AppendFormatted Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

AppendFormatted(ReadOnlySpan<Char>)	Writes the specified value to the handler.
AppendFormatted(String)	Writes the specified value to the handler.
AppendFormatted(Object, Int32, String)	Writes the specified value to the handler.
AppendFormatted(ReadOnlySpan<Char>, Int32, String)	Writes the specified value to the handler.
AppendFormatted(String, Int32, String)	Writes a specified value to the handler using a specified format string.
AppendFormatted<T>(T)	Writes the specified value to the handler.
AppendFormatted<T>(T, Int32)	Writes the specified value to the handler.
AppendFormatted<T>(T, String)	Writes a specified value to the handler using a specified format string.
AppendFormatted<T>(T, Int32, String)	Writes a specified value to the handler using a specified format string.

AppendFormatted(ReadOnlySpan<Char>)

Writes the specified value to the handler.

```
C#
```

```
public bool AppendFormatted (scoped ReadOnlySpan<char> value);
```

Parameters

`value` ReadOnlySpan<Char>

The value to write.

Returns

Boolean

`false` if the operation failed; `true` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted(String)

Writes the specified value to the handler.

C#

```
public bool AppendFormatted (string? value);
```

Parameters

`value` String

The value to write.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted(Object, Int32, String)

Writes the specified value to the handler.

```
C#

public bool AppendFormatted (object? value, int alignment = 0, string?
format = default);
```

Parameters

`value` Object

The value to write.

`alignment` Int32

The minimum number of characters that should be written for this value. A negative value indicates left-aligned, and the required minimum is the absolute value.

`format` String

The format string.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted(ReadOnlySpan<Char>, Int32, String)

Writes the specified value to the handler.

```
C#

public bool AppendFormatted (scoped ReadOnlySpan<char> value, int
alignment = 0, string? format = default);
```

Parameters

`value` ReadOnlySpan<Char>

The value to write.

`alignment` Int32

>The minimum number of characters that should be written for this value. If the value is negative, it indicates left-aligned and the required minimum is the absolute value.

`format` String

The format string.

Returns

Boolean

`false` if the operation failed; `true` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted(String, Int32, String)

Writes a specified value to the handler using a specified format string.

```
C#

public bool AppendFormatted (string? value, int alignment = 0, string?
format = default);
```

Parameters

`value` String

The value to write.

`alignment` Int32

The minimum number of characters that should be written for this value. A negative value indicates left-aligned, and the required minimum is the absolute value.

`format` String

The format string.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted<T>(T)

Writes the specified value to the handler.

```
C#
```

```
public bool AppendFormatted<T> (T value);
```

Type Parameters

T

The type of the value to write.

Parameters

value T

The value to write.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted<T>(T, Int32)

Writes the specified value to the handler.

C#

```
public bool AppendFormatted<T> (T value, int alignment);
```

Type Parameters

T

The type of the value to write.

Parameters

`value` T

The value to write.

`alignment` Int32

The minimum number of characters that should be written for this value.A negative value indicates left-aligned, and the required minimum is the absolute value.

Returns

Boolean

`false` if the operation failed; `true` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted<T>(T, String)

Writes a specified value to the handler using a specified format string.

```
C#
public bool AppendFormatted<T> (T value, string? format);
```

Type Parameters

`T`

The type of the value to write.

Parameters

`value` T

The value to write.

`format` String

The format string.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AppendFormatted<T>(T, Int32, String)

Writes a specified value to the handler using a specified format string.

```
C#
public bool AppendFormatted<T> (T value, int alignment, string? format);
```

Type Parameters

T

The type of the value to write.

Parameters

value T

The value to write.

alignment Int32

The minimum number of characters that should be written for this value. A negative value indicates left-aligned, and the required minimum is the absolute value.

format String

The format string.

Returns

Boolean

`true` on success; `false` otherwise.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MemoryExtensions.TryWriteInterpolatedStringHandler.AppendLiteral Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Writes the specified string to the handler.

C#

```
public bool AppendLiteral (string value);
```

Parameters

`value` String

The string to write.

Returns

Boolean

`true` if the value could be formatted to the span; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

MethodAccessException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an invalid attempt to access a method, such as accessing a private method from partially trusted code.

C#

```
public class MethodAccessException : MemberAccessException
```

Inheritance Object → Exception → SystemException → MemberAccessException → MethodAccessException

Remarks

> **Note**
>
> This exception is not included in the **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **MemberAccessException** instead.

This exception is thrown in situations such as the following:

- A private, protected, or internal method that would not be accessible from normal compiled code is accessed from partially trusted code by using reflection.
- A security-critical method is accessed from transparent code.
- The access level of a method in a class library has changed, and one or more assemblies that reference the library have not been recompiled.

> **Note**

Beginning with the .NET Framework 4, the common language runtime treats application code as transparent when it is run with partial trust. See **Security Considerations for Reflection**.

MethodAccessException uses the HRESULT COR_E_METHODACCESS, that has the value 0x80131510.

For a list of initial property values for an instance of MethodAccessException, see the MethodAccessException constructors.

Constructors

Expand table

MethodAccessException()	Initializes a new instance of the MethodAccessException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "Attempt to access the method failed." This message takes into account the current system culture.
MethodAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MethodAccessException class with serialized data.
MethodAccessException(String)	Initializes a new instance of the MethodAccessException class with a specified error message.
MethodAccessException(String, Exception)	Initializes a new instance of the MethodAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

	(Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MethodAccessException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MethodAccessException class.

Overloads

Expand table

MethodAccessException()	Initializes a new instance of the MethodAccessException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "Attempt to access the method failed." This message takes into account the current system culture.
MethodAccessException(String)	Initializes a new instance of the MethodAccessException class with a specified error message.
MethodAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MethodAccessException class with serialized data.
MethodAccessException(String, Exception)	Initializes a new instance of the MethodAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

MethodAccessException()

Initializes a new instance of the MethodAccessException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "Attempt to access the method failed." This message takes into account the current system culture.

```
C#
public MethodAccessException ();
```

Remarks

The following table shows the initial property values for an instance of MethodAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MethodAccessException(String)

Initializes a new instance of the MethodAccessException class with a specified error message.

```
C#
public MethodAccessException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of `message` should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MethodAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MethodAccessException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the MethodAccessException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected MethodAccessException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MethodAccessException(String, Exception)

Initializes a new instance of the MethodAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public MethodAccessException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MethodAccessException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MidpointRounding Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the strategy that mathematical rounding methods should use to round a number.

C#

```
public enum MidpointRounding
```

Inheritance Object → ValueType → Enum → MidpointRounding

Fields

Expand table

AwayFromZero	1	The strategy of rounding to the nearest number, and when a number is halfway between two others, it's rounded toward the nearest number that's away from zero.
ToEven	0	The strategy of rounding to the nearest number, and when a number is halfway between two others, it's rounded toward the nearest even number.
ToNegativeInfinity	3	The strategy of downwards-directed rounding, with the result closest to and no greater than the infinitely precise result.
ToPositiveInfinity	4	The strategy of upwards-directed rounding, with the result closest to and no less than the infinitely precise result.
ToZero	2	The strategy of directed rounding toward zero, with the result closest to and no greater in magnitude than the infinitely precise result.

Examples

The following example demonstrates the Math.Round method in conjunction with the `MidpointRounding` enumeration:

C#

```
decimal result;

// Round a positive value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
MidpointRounding.ToEven)");
result = Math.Round(3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(3.47m, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result} = Math.Round({3.47m}, 1,
MidpointRounding.ToZero)\n");

// Round a negative value using different strategies.
// The precision of the result is 1 decimal place.

result = Math.Round(-3.45m, 1, MidpointRounding.ToEven);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.ToEven)");
result = Math.Round(-3.45m, 1, MidpointRounding.AwayFromZero);
Console.WriteLine($"{result} = Math.Round({-3.45m}, 1,
MidpointRounding.AwayFromZero)");
result = Math.Round(-3.47m, 1, MidpointRounding.ToZero);
Console.WriteLine($"{result} = Math.Round({-3.47m}, 1,
MidpointRounding.ToZero)\n");

/*
This code example produces the following results:

3.4 = Math.Round(3.45, 1, MidpointRounding.ToEven)
3.5 = Math.Round(3.45, 1, MidpointRounding.AwayFromZero)
3.4 = Math.Round(3.47, 1, MidpointRounding.ToZero)

-3.4 = Math.Round(-3.45, 1, MidpointRounding.ToEven)
-3.5 = Math.Round(-3.45, 1, MidpointRounding.AwayFromZero)
-3.4 = Math.Round(-3.47, 1, MidpointRounding.ToZero)
*/
```

Remarks

For more information about this API, see Supplemental API remarks for MidpointRounding.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingFieldException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to dynamically access a field that does not exist. If a field in a class library has been removed or renamed, recompile any assemblies that reference that library.

C#

```
public class MissingFieldException : MissingMemberException
```

Inheritance Object → Exception → SystemException → MemberAccessException → MissingMemberException → MissingFieldException

Examples

This example shows what happens if you attempt to use reflection to call a method that does not exist and access a field that does not exist. The application recovers by catching the MissingMethodException, MissingFieldException, and MissingMemberException.

C#

```
using System;
using System.Reflection;

public class App
{
    public static void Main()
    {

        try
        {
            // Attempt to call a static DoSomething method defined in the
App class.
            // However, because the App class does not define this method,
            // a MissingMethodException is thrown.
            typeof(App).InvokeMember("DoSomething", BindingFlags.Static |
```

```
                BindingFlags.InvokeMethod, null, null, null);
        }
        catch (MissingMethodException e)
        {
            // Show the user that the DoSomething method cannot be called.
            Console.WriteLine("Unable to call the DoSomething method: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AField field defined in the App
class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AField", BindingFlags.Static |
BindingFlags.SetField,
                null, null, new Object[] { 5 });
        }
        catch (MissingFieldException e)
        {
         // Show the user that the AField field cannot be accessed.
         Console.WriteLine("Unable to access the AField field: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AnotherField field defined in the
App class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AnotherField", BindingFlags.Static |
                BindingFlags.GetField, null, null, null);
        }
        catch (MissingMemberException e)
        {
         // Notice that this code is catching MissingMemberException which
is the
         // base class of MissingMethodException and MissingFieldException.
         // Show the user that the AnotherField field cannot be accessed.
         Console.WriteLine("Unable to access the AnotherField field: {0}",
e.Message);
        }
    }
}
// This code example produces the following output:
//
// Unable to call the DoSomething method: Method 'App.DoSomething' not
found.
// Unable to access the AField field: Field 'App.AField' not found.
// Unable to access the AnotherField field: Field 'App.AnotherField' not
found.
```

Remarks

Normally a compilation error is generated if code attempts to access a nonexistent member of a class. MissingFieldException is designed to handle cases where an attempt is made to dynamically access a renamed or deleted field of an assembly that is not referenced by its strong name. The MissingFieldException is thrown when code in a dependent assembly attempts to access a missing field in an assembly that was modified.

MissingFieldException uses the HRESULT COR_E_MISSINGFIELD, that has the value 0x80131511.

For a list of initial property values for an instance of MissingFieldException, see the MissingFieldException constructors.

Constructors

Expand table

MissingFieldException()	Initializes a new instance of the MissingFieldException class.
MissingFieldException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingFieldException class with serialized data.
MissingFieldException(String)	Initializes a new instance of the MissingFieldException class with a specified error message.
MissingFieldException(String, Exception)	Initializes a new instance of the MissingFieldException class with a specified error message and a reference to the inner exception that is the cause of this exception.
MissingFieldException(String, String)	Initializes a new instance of the MissingFieldException class with the specified class name and field name.

Fields

Expand table

ClassName	Holds the class name of the missing member. (Inherited from MissingMemberException)
MemberName	Holds the name of the missing member.

	(Inherited from MissingMemberException)
Signature	Holds the signature of the missing member. (Inherited from MissingMemberException)

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the text string showing the signature of the missing field, the class name, and the field name. This property is read-only.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)

GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, the member name, the signature of the missing member, and additional exception information. (Inherited from MissingMemberException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception

- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingFieldException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MissingFieldException class.

Overloads

Expand table

MissingFieldException()	Initializes a new instance of the MissingFieldException class.
MissingFieldException(String)	Initializes a new instance of the MissingFieldException class with a specified error message.
MissingFieldException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingFieldException class with serialized data.
MissingFieldException(String, Exception)	Initializes a new instance of the MissingFieldException class with a specified error message and a reference to the inner exception that is the cause of this exception.
MissingFieldException(String, String)	Initializes a new instance of the MissingFieldException class with the specified class name and field name.

MissingFieldException()

Initializes a new instance of the MissingFieldException class.

C#

```
public MissingFieldException ();
```

Remarks

This constructor initializes Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to access a non-existing field." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MissingFieldException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingFieldException(String)

Initializes a new instance of the MissingFieldException class with a specified error message.

C#

```
public MissingFieldException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of `message` should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MissingFieldException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingFieldException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the MissingFieldException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected MissingFieldException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MissingFieldException(String, Exception)

Initializes a new instance of the MissingFieldException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public MissingFieldException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MissingFieldException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingFieldException(String, String)

Initializes a new instance of the MissingFieldException class with the specified class name and field name.

C#

```
public MissingFieldException (string? className, string? fieldName);
```

Parameters

`className` String

The name of the class in which access to a nonexistent field was attempted.

`fieldName` String

The name of the field that cannot be accessed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingFieldException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the text string showing the signature of the missing field, the class name, and the field name. This property is read-only.

C#

```
public override string Message { get; }
```

Property Value

String

The error message string.

Examples

The following example demonstrates the Message property. This code example is part of a larger example provided for the MissingFieldException class.

C#

```
try
{
    // Attempt to access a static AField field defined in the App class.
    // However, because the App class does not define this field,
    // a MissingFieldException is thrown.
    typeof(App).InvokeMember("AField", BindingFlags.Static |
BindingFlags.SetField,
        null, null, new Object[] { 5 });
}
catch (MissingFieldException e)
{
 // Show the user that the AField field cannot be accessed.
```

```
  Console.WriteLine("Unable to access the AField field: {0}", e.Message);
 }
```

Remarks

If the class name is not specified when the object is constructed, the default text string inherited from the base class is returned. This property overrides Message. The error message should be localized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to dynamically access a class member that does not exist or that is not declared as public. If a member in a class library has been removed or renamed, recompile any assemblies that reference that library.

C#

```
public class MissingMemberException : MemberAccessException
```

Inheritance Object → Exception → SystemException → MemberAccessException → MissingMemberException

Derived System.MissingFieldException
System.MissingMethodException

Examples

This example shows what happens if you attempt to use reflection to call a method that does not exist and access a field that does not exist. The application recovers by catching the MissingMethodException, MissingFieldException, and MissingMemberException.

C#

```
using System;
using System.Reflection;

public class App
{
    public static void Main()
    {

        try
        {
```

```
            // Attempt to call a static DoSomething method defined in the
App class.
            // However, because the App class does not define this method,
            // a MissingMethodException is thrown.
            typeof(App).InvokeMember("DoSomething", BindingFlags.Static |
                BindingFlags.InvokeMethod, null, null, null);
        }
        catch (MissingMethodException e)
        {
            // Show the user that the DoSomething method cannot be called.
            Console.WriteLine("Unable to call the DoSomething method: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AField field defined in the App
class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AField", BindingFlags.Static |
BindingFlags.SetField,
                null, null, new Object[] { 5 });
        }
        catch (MissingFieldException e)
        {
         // Show the user that the AField field cannot be accessed.
         Console.WriteLine("Unable to access the AField field: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AnotherField field defined in the
App class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AnotherField", BindingFlags.Static |
                BindingFlags.GetField, null, null, null);
        }
        catch (MissingMemberException e)
        {
         // Notice that this code is catching MissingMemberException which
is the
         // base class of MissingMethodException and MissingFieldException.
         // Show the user that the AnotherField field cannot be accessed.
         Console.WriteLine("Unable to access the AnotherField field: {0}",
e.Message);
        }
    }
}
// This code example produces the following output:
//
// Unable to call the DoSomething method: Method 'App.DoSomething' not
found.
```

```
// Unable to access the AField field: Field 'App.AField' not found.
// Unable to access the AnotherField field: Field 'App.AnotherField' not
found.
```

Remarks

Normally a compilation error is generated if the code attempts to access a nonexistent member of a class. MissingMemberException is designed to handle cases where a field or a method is deleted or renamed in one assembly and the change is not reflected in a second assembly. At runtime, MissingMemberException will be thrown when the code in the second assembly attempts to access the missing member in the first assembly.

MissingMemberException is the base class for MissingFieldException and MissingMethodException. In general it is better to use one of the derived classes of MissingMemberException to more precisely indicate the exact nature of the error. Throw a MissingMemberException if you are only interested in capturing the general case of a missing member error.

MissingMemberException uses the HRESULT COR_E_MISSINGMEMBER, that has the value 0x80131512.

For a list of initial property values for an instance of MissingMemberException, see the MissingMemberException constructors.

Constructors

Expand table

MissingMemberException()	Initializes a new instance of the MissingMemberException class.
MissingMember Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingMemberException class with serialized data.
MissingMemberException(String)	Initializes a new instance of the MissingMemberException class with a specified error message.
MissingMemberException(String, Exception)	Initializes a new instance of the MissingMemberException class with a specified error message and a reference to the inner exception that is the root cause of this exception.
MissingMemberException(String,	Initializes a new instance of the MissingMemberException

String)	class with the specified class name and member name.

Fields

Expand table

ClassName	Holds the class name of the missing member.
MemberName	Holds the name of the missing member.
Signature	Holds the signature of the missing member.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the text string showing the class name, the member name, and the signature of the missing member.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, the member name, the signature of the missing member, and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- MissingFieldException
- MissingMethodException
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MissingMemberException class.

Overloads

Expand table

MissingMemberException()	Initializes a new instance of the MissingMemberException class.
MissingMemberException(String)	Initializes a new instance of the MissingMemberException class with a specified error message.
MissingMemberException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingMemberException class with serialized data.
MissingMemberException(String, Exception)	Initializes a new instance of the MissingMemberException class with a specified error message and a reference to the inner exception that is the root cause of this exception.
MissingMemberException(String, String)	Initializes a new instance of the MissingMemberException class with the specified class name and member name.

MissingMemberException()

Initializes a new instance of the MissingMemberException class.

```
C#
```

```
public MissingMemberException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to access a missing member." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MissingMemberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMemberException(String)

Initializes a new instance of the MissingMemberException class with a specified error message.

C#

```
public MissingMemberException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter should be understandable by the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MissingMemberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMemberException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the MissingMemberException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected MissingMemberException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MissingMemberException(String, Exception)

Initializes a new instance of the MissingMemberException class with a specified error message and a reference to the inner exception that is the root cause of this exception.

C#

```
public MissingMemberException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

An instance of Exception that is the cause of the current `Exception`. If `inner` is not a null reference (`Nothing` in Visual Basic), then the current `Exception` is raised in a catch block handling `inner`.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MissingMemberException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMemberException(String, String)

Initializes a new instance of the MissingMemberException class with the specified class name and member name.

C#

```
public MissingMemberException (string? className, string? memberName);
```

Parameters

`className` String

The name of the class in which access to a nonexistent member was attempted.

`memberName` String

The name of the member that cannot be accessed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException.ClassName Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Holds the class name of the missing member.

C#

```
protected string? ClassName;
```

Field Value

String

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException.MemberName Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Holds the name of the missing member.

C#

```
protected string? MemberName;
```

Field Value

String

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException.Signature Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Holds the signature of the missing member.

C#

```
protected byte[]? Signature;
```

Field Value

Byte[]

Remarks

Signature contains a Byte value that represents the signature of the missing member.

Format is used to format the signature. For additional information, see SignatureHelper, SignatureToken, and String.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SignatureHelper

- SignatureToken

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the text string showing the class name, the member name, and the signature of the missing member.

C#

```
public override string Message { get; }
```

Property Value

String

The error message string.

Examples

The following example demonstrates the Message property. This code example is part of a larger example provided for the MissingMemberException class.

C#

```
try
{
    // Attempt to access a static AnotherField field defined in the App
class.
    // However, because the App class does not define this field,
    // a MissingFieldException is thrown.
    typeof(App).InvokeMember("AnotherField", BindingFlags.Static |
        BindingFlags.GetField, null, null, null);
}
catch (MissingMemberException e)
{
 // Notice that this code is catching MissingMemberException which is the
 // base class of MissingMethodException and MissingFieldException.
 // Show the user that the AnotherField field cannot be accessed.
```

```
 Console.WriteLine("Unable to access the AnotherField field: {0}",
e.Message);
}
```

Remarks

If the class name is not specified when the object is constructed, the default text string inherited from the base class is returned.

This property overrides Message. The error message should be localized.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- MissingFieldException
- MissingMethodException
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMemberException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the class name, the member name, the signature of the missing member, and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is `null`.

Remarks

GetObjectData sets a SerializationInfo with all the exception object data targeted for serialization. During deserialization, the exception object is reconstituted from the SerializationInfo transmitted over the stream.

For more information, see SerializationInfo.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMethodException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to dynamically access a method that does not exist.

C#

```
public class MissingMethodException : MissingMemberException
```

Inheritance Object → Exception → SystemException → MemberAccessException → MissingMemberException → MissingMethodException

Examples

This example shows what happens if you attempt to use reflection to call a method that does not exist and access a field that does not exist. The application recovers by catching the MissingMethodException, MissingFieldException, and MissingMemberException.

C#

```
using System;
using System.Reflection;

public class App
{
    public static void Main()
    {

        try
        {
            // Attempt to call a static DoSomething method defined in the
App class.
            // However, because the App class does not define this method,
            // a MissingMethodException is thrown.
            typeof(App).InvokeMember("DoSomething", BindingFlags.Static |
                BindingFlags.InvokeMethod, null, null, null);
        }
```

```
        catch (MissingMethodException e)
        {
            // Show the user that the DoSomething method cannot be called.
            Console.WriteLine("Unable to call the DoSomething method: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AField field defined in the App
class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AField", BindingFlags.Static |
BindingFlags.SetField,
                null, null, new Object[] { 5 });
        }
        catch (MissingFieldException e)
        {
         // Show the user that the AField field cannot be accessed.
         Console.WriteLine("Unable to access the AField field: {0}",
e.Message);
        }

        try
        {
            // Attempt to access a static AnotherField field defined in the
App class.
            // However, because the App class does not define this field,
            // a MissingFieldException is thrown.
            typeof(App).InvokeMember("AnotherField", BindingFlags.Static |
                BindingFlags.GetField, null, null, null);
        }
        catch (MissingMemberException e)
        {
         // Notice that this code is catching MissingMemberException which
is the
         // base class of MissingMethodException and MissingFieldException.
         // Show the user that the AnotherField field cannot be accessed.
         Console.WriteLine("Unable to access the AnotherField field: {0}",
e.Message);
        }
    }
}
// This code example produces the following output:
//
// Unable to call the DoSomething method: Method 'App.DoSomething' not
found.
// Unable to access the AField field: Field 'App.AField' not found.
// Unable to access the AnotherField field: Field 'App.AnotherField' not
found.
```

Remarks

Normally a compilation error is generated if code attempts to access a nonexistent method of a class. MissingMethodException is designed to handle cases where an attempt is made to dynamically access a renamed or deleted method of an assembly that is not referenced by its strong name. MissingMethodException is thrown when code in a dependent assembly attempts to access a missing method in an assembly that was modified.

MissingMethodException uses the HRESULT COR_E_MISSINGMETHOD, that has the value 0x80131513.

For a list of initial property values for an instance of MissingMethodException, see the MissingMethodException constructors.

The exact timing of when statically referenced methods are loaded is unspecified. This exception may be thrown before the method that references the missing method starts executing.

> **Note**
>
> This exception is not included in **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **MissingMemberException** instead.

Constructors

Expand table

MissingMethodException()	Initializes a new instance of the MissingMethodException class.
MissingMethodException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingMethodException class with serialized data.
MissingMethodException(String)	Initializes a new instance of the MissingMethodException class with a specified error message.
MissingMethodException(String, Exception)	Initializes a new instance of the MissingMethodException class with a specified error message and a reference to the inner exception that is the cause of this exception.

MissingMethodException(String, String)	Initializes a new instance of the MissingMethodException class with the specified class name and method name.

Fields

Expand table

ClassName	Holds the class name of the missing member. (Inherited from MissingMemberException)
MemberName	Holds the name of the missing member. (Inherited from MissingMemberException)
Signature	Holds the signature of the missing member. (Inherited from MissingMemberException)

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the text string showing the class name, the method name, and the signature of the missing method. This property is read-only.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, the member name, the signature of the missing member, and additional exception information. (Inherited from MissingMemberException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMethodException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MissingMethodException class.

Overloads

Expand table

MissingMethodException()	Initializes a new instance of the MissingMethodException class.
MissingMethodException(String)	Initializes a new instance of the MissingMethodException class with a specified error message.
MissingMethodException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the MissingMethodException class with serialized data.
MissingMethodException(String, Exception)	Initializes a new instance of the MissingMethodException class with a specified error message and a reference to the inner exception that is the cause of this exception.
MissingMethodException(String, String)	Initializes a new instance of the MissingMethodException class with the specified class name and method name.

MissingMethodException()

Initializes a new instance of the MissingMethodException class.

```
C#
```

```
public MissingMethodException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to access a missing method." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MissingMethodException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMethodException(String)

Initializes a new instance of the MissingMethodException class with a specified error message.

C#

```
public MissingMethodException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of the `message` parameter should be understandable by the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MissingMethodException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMethodException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the MissingMethodException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected MissingMethodException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MissingMethodException(String, Exception)

Initializes a new instance of the MissingMethodException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public MissingMethodException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MissingMethodException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MissingMethodException(String, String)

Initializes a new instance of the MissingMethodException class with the specified class name and method name.

C#

```
public MissingMethodException (string? className, string? methodName);
```

Parameters

`className` String

The name of the class in which access to a nonexistent method was attempted.

`methodName` String

The name of the method that cannot be accessed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MissingMethodException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the text string showing the class name, the method name, and the signature of the missing method. This property is read-only.

C#

```
public override string Message { get; }
```

Property Value

String

The error message string.

Examples

The following example demonstrates the Message property. This code example is part of a larger example provided for the MissingMethodException class.

C#

```
try
{
    // Attempt to call a static DoSomething method defined in the App class.
    // However, because the App class does not define this method,
    // a MissingMethodException is thrown.
    typeof(App).InvokeMember("DoSomething", BindingFlags.Static |
        BindingFlags.InvokeMethod, null, null, null);
}
catch (MissingMethodException e)
{
    // Show the user that the DoSomething method cannot be called.
    Console.WriteLine("Unable to call the DoSomething method: {0}",
```

```
e.Message);
}
```

Remarks

If the class name is not specified when the object is constructed, the default text string inherited from the base class is returned.

This property overrides Message. The error message should be localized.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ModuleHandle Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a runtime handle for a module.

C#

```
public struct ModuleHandle : IEquatable<ModuleHandle>
```

Inheritance Object → ValueType → ModuleHandle

Implements IEquatable<ModuleHandle>

Remarks

The ResolveFieldHandle, ResolveMethodHandle, and ResolveTypeHandle methods provide efficient resolution of metadata tokens to runtime handles for fields, methods, and types in the current module.

Fields

Expand table

EmptyHandle	Represents an empty module handle.

Properties

Expand table

MDStreamVersion	Gets the metadata stream version.

Methods

Expand table

Equals(ModuleHandle)	Returns a Boolean value indicating whether the specified ModuleHandle structure is equal to the current ModuleHandle.
Equals(Object)	Returns a Boolean value indicating whether the specified object is a ModuleHandle structure, and equal to the current ModuleHandle.
GetHashCode()	Returns the hash code for this instance.
GetRuntimeFieldHandleFromMetadataToken(Int32)	Returns a runtime handle for the field identified by the specified metadata token.
GetRuntimeMethodHandleFromMetadataToken(Int32)	Returns a runtime method handle for the method or constructor identified by the specified metadata token.
GetRuntimeTypeHandleFromMetadataToken(Int32)	Returns a runtime type handle for the type identified by the specified metadata token.
ResolveFieldHandle(Int32)	Returns a runtime handle for the field identified by the specified metadata token.
ResolveFieldHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime field handle for the field identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.
ResolveMethodHandle(Int32)	Returns a runtime method handle for the method or constructor identified by the specified metadata token.
ResolveMethodHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime method handle for the method or constructor identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.
ResolveTypeHandle(Int32)	Returns a runtime type handle for the type identified by the specified metadata token.
ResolveTypeHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime type handle for the type identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

Operators

Expand table

Equality(ModuleHandle, ModuleHandle)	Tests whether two ModuleHandle structures are equal.
Inequality(ModuleHandle, ModuleHandle)	Tests whether two ModuleHandle structures are unequal.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Module

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ModuleHandle.EmptyHandle Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an empty module handle.

C#

```
public static readonly ModuleHandle EmptyHandle;
```

Field Value

ModuleHandle

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ModuleHandle.MDStreamVersion Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the metadata stream version.

```
C#
public int MDStreamVersion { get; }
```

Property Value

Int32

A 32-bit integer representing the metadata stream version. The high-order two bytes represent the major version number, and the low-order two bytes represent the minor version number.

Remarks

> **Note**
>
> For more information on the metadata header, see the metadata specification in Partition II of **ECMA-335 Common Language Infrastructure (CLI)**.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ModuleHandle.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether a specified ModuleHandle structure is equal to the current ModuleHandle.

Overloads

Expand table

Equals(ModuleHandle)	Returns a Boolean value indicating whether the specified ModuleHandle structure is equal to the current ModuleHandle.
Equals(Object)	Returns a Boolean value indicating whether the specified object is a ModuleHandle structure, and equal to the current ModuleHandle.

Equals(ModuleHandle)

Returns a Boolean value indicating whether the specified ModuleHandle structure is equal to the current ModuleHandle.

C#

```
public bool Equals (ModuleHandle handle);
```

Parameters

`handle` ModuleHandle

The ModuleHandle structure to be compared with the current ModuleHandle.

Returns

Boolean

`true` if `handle` is equal to the current ModuleHandle structure; otherwise `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(Object)

Returns a Boolean value indicating whether the specified object is a ModuleHandle structure, and equal to the current ModuleHandle.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to be compared with the current ModuleHandle structure.

Returns

Boolean

`true` if `obj` is a ModuleHandle structure, and is equal to the current ModuleHandle structure; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ModuleHandle.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that is the hash code for this instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetHashCode()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ModuleHandle.GetRuntimeFieldHandleFromMetadataToken(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime handle for the field identified by the specified metadata token.

C#

```
public RuntimeFieldHandle GetRuntimeFieldHandleFromMetadataToken (int
fieldToken);
```

Parameters

`fieldToken` Int32

A metadata token that identifies a field in the module.

Returns

RuntimeFieldHandle

A RuntimeFieldHandle for the field identified by `fieldToken`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ModuleHandle.GetRuntimeMethodHandleFromMetadataToken(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime method handle for the method or constructor identified by the specified metadata token.

C#

```
public RuntimeMethodHandle GetRuntimeMethodHandleFromMetadataToken (int methodToken);
```

Parameters

`methodToken` Int32

A metadata token that identifies a method or constructor in the module.

Returns

RuntimeMethodHandle

A RuntimeMethodHandle for the method or constructor identified by `methodToken`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ModuleHandle.GetRuntimeTypeHandleFromMetadataToken(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime type handle for the type identified by the specified metadata token.

C#

```
public RuntimeTypeHandle GetRuntimeTypeHandleFromMetadataToken (int
typeToken);
```

Parameters

typeToken Int32

A metadata token that identifies a type in the module.

Returns

RuntimeTypeHandle

A RuntimeTypeHandle for the type identified by `typeToken`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ModuleHandle.ResolveFieldHandle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime field handle for a field identified by a metadata token.

Overloads

Expand table

ResolveFieldHandle(Int32)	Returns a runtime handle for the field identified by the specified metadata token.
ResolveFieldHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime field handle for the field identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

ResolveFieldHandle(Int32)

Returns a runtime handle for the field identified by the specified metadata token.

C#

```
public RuntimeFieldHandle ResolveFieldHandle (int fieldToken);
```

Parameters

`fieldToken` Int32

A metadata token that identifies a field in the module.

Returns

RuntimeFieldHandle

A RuntimeFieldHandle for the field identified by `fieldToken`.

Exceptions

BadImageFormatException

`metadataToken` is not a valid token in the scope of the current module.

-or-

`metadataToken` is not a token for a field in the scope of the current module.

-or-

`metadataToken` identifies a field whose parent `TypeSpec` has a signature containing element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty field handle.

Remarks

To resolve a metadata token that identifies a field whose parent `TypeSpec` has a signature containing element type `var` or `mvar`, use the ResolveFieldHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[]) method overload, which allows you to supply the necessary context.

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

ResolveFieldHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])

Returns a runtime field handle for the field identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

C#

```
public RuntimeFieldHandle ResolveFieldHandle (int fieldToken,
RuntimeTypeHandle[]? typeInstantiationContext, RuntimeTypeHandle[]?
methodInstantiationContext);
```

Parameters

`fieldToken` Int32

A metadata token that identifies a field in the module.

`typeInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures representing the generic type arguments of the type where the token is in scope, or `null` if that type is not generic.

`methodInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures representing the generic type arguments of the method where the token is in scope, or `null` if that method is not generic.

Returns

RuntimeFieldHandle

A RuntimeFieldHandle for the field identified by `fieldToken`.

Exceptions

BadImageFormatException

`metadataToken` is not a valid token in the scope of the current module.

-or-

`metadataToken` is not a token for a field in the scope of the current module.

-or-

`metadataToken` identifies a field whose parent `TypeSpec` has a signature containing element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty field handle.

ArgumentOutOfRangeException

`fieldToken` is not a valid token.

Remarks

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ModuleHandle.ResolveMethodHandle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime method handle for a method or constructor identified by a metadata token.

Overloads

Expand table

ResolveMethodHandle(Int32)	Returns a runtime method handle for the method or constructor identified by the specified metadata token.
ResolveMethodHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime method handle for the method or constructor identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

ResolveMethodHandle(Int32)

Returns a runtime method handle for the method or constructor identified by the specified metadata token.

C#

```
public RuntimeMethodHandle ResolveMethodHandle (int methodToken);
```

Parameters

`methodToken` Int32

A metadata token that identifies a method or constructor in the module.

Returns

RuntimeMethodHandle

A RuntimeMethodHandle for the method or constructor identified by `methodToken`.

Exceptions

BadImageFormatException

`methodToken` is not a valid metadata token for a method in the current module.

-or-

`metadataToken` is not a token for a method or constructor in the scope of the current module.

-or-

`metadataToken` is a `MethodSpec` whose signature contains element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty method handle.

Remarks

To resolve a metadata token for a `MethodSpec` whose signature contains element type `var` or `mvar`, use the ResolveMethodHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[]) method overload, which allows you to supply the necessary context.

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ResolveMethodHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])

Returns a runtime method handle for the method or constructor identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

C#

```
public RuntimeMethodHandle ResolveMethodHandle (int methodToken,
RuntimeTypeHandle[]? typeInstantiationContext, RuntimeTypeHandle[]?
methodInstantiationContext);
```

Parameters

`methodToken` Int32

A metadata token that identifies a method or constructor in the module.

`typeInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures representing the generic type arguments of the type where the token is in scope, or `null` if that type is not generic.

`methodInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures representing the generic type arguments of the method where the token is in scope, or `null` if that method is not generic.

Returns

RuntimeMethodHandle

A RuntimeMethodHandle for the method or constructor identified by `methodToken`.

Exceptions

BadImageFormatException

`methodToken` is not a valid metadata token for a method in the current module.

-or-

`metadataToken` is not a token for a method or constructor in the scope of the current module.

-or-

`metadataToken` is a `MethodSpec` whose signature contains element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty method handle.

ArgumentOutOfRangeException

`methodToken` is not a valid token.

Remarks

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

ModuleHandle.ResolveTypeHandle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a runtime type handle for a type identified by a metadata token.

Overloads

Expand table

ResolveTypeHandle(Int32)	Returns a runtime type handle for the type identified by the specified metadata token.
ResolveTypeHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])	Returns a runtime type handle for the type identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

ResolveTypeHandle(Int32)

Returns a runtime type handle for the type identified by the specified metadata token.

C#

```
public RuntimeTypeHandle ResolveTypeHandle (int typeToken);
```

Parameters

`typeToken` Int32

A metadata token that identifies a type in the module.

Returns

RuntimeTypeHandle

A RuntimeTypeHandle for the type identified by `typeToken`.

Exceptions

BadImageFormatException

`typeToken` is not a valid metadata token for a type in the current module.

-or-

`metadataToken` is not a token for a type in the scope of the current module.

-or-

`metadataToken` is a `TypeSpec` whose signature contains element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty type handle.

Remarks

To resolve a metadata token for a `TypeSpec` whose signature contains element type `var` or `mvar`, use the ResolveTypeHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[]) method overload, which allows you to supply the necessary context.

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

ResolveTypeHandle(Int32, RuntimeTypeHandle[], RuntimeTypeHandle[])

Returns a runtime type handle for the type identified by the specified metadata token, specifying the generic type arguments of the type and method where the token is in scope.

C#

```
public RuntimeTypeHandle ResolveTypeHandle (int typeToken,
RuntimeTypeHandle[]? typeInstantiationContext, RuntimeTypeHandle[]?
methodInstantiationContext);
```

Parameters

`typeToken` Int32

A metadata token that identifies a type in the module.

`typeInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures representing the generic type arguments of the type where the token is in scope, or `null` if that type is not generic.

`methodInstantiationContext` RuntimeTypeHandle[]

An array of RuntimeTypeHandle structures objects representing the generic type arguments of the method where the token is in scope, or `null` if that method is not generic.

Returns

RuntimeTypeHandle

A RuntimeTypeHandle for the type identified by `typeToken`.

Exceptions

BadImageFormatException

`typeToken` is not a valid metadata token for a type in the current module.

-or-

`metadataToken` is not a token for a type in the scope of the current module.

-or-

`metadataToken` is a `TypeSpec` whose signature contains element type `var` or `mvar`.

InvalidOperationException

The method is called on an empty type handle.

ArgumentOutOfRangeException

`typeToken` is not a valid token.

Remarks

> **Note**
>
> Information about metadata tokens can be found in the **Common Language Infrastructure (CLI) documentation**, especially "Partition II: Metadata Definition and Semantics".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ModuleHandle.Equality(ModuleHandle, ModuleHandle) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tests whether two ModuleHandle structures are equal.

```
C#
public static bool operator == (ModuleHandle left, ModuleHandle right);
```

Parameters

`left` ModuleHandle

The ModuleHandle structure to the left of the equality operator.

`right` ModuleHandle

The ModuleHandle structure to the right of the equality operator.

Returns

Boolean

`true` if the ModuleHandle structures are equal; otherwise, `false`.

Remarks

The equivalent method for this operator is ModuleHandle.Equals(ModuleHandle).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ModuleHandle.Inequality(ModuleHandle, ModuleHandle) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tests whether two ModuleHandle structures are unequal.

```
C#
public static bool operator != (ModuleHandle left, ModuleHandle right);
```

Parameters

`left` ModuleHandle

The ModuleHandle structure to the left of the inequality operator.

`right` ModuleHandle

The ModuleHandle structure to the right of the inequality operator.

Returns

Boolean

`true` if the ModuleHandle structures are unequal; otherwise, `false`.

Remarks

The equivalent method for this operator is ModuleHandle.Equals(ModuleHandle).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MTAThreadAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that the COM threading model for an application is multithreaded apartment (MTA).

C#

```
[System.AttributeUsage(System.AttributeTargets.Method)]
public sealed class MTAThreadAttribute : Attribute
```

Inheritance Object → Attribute → MTAThreadAttribute

Attributes AttributeUsageAttribute

Remarks

Apply this attribute to the entry point method (the `Main()` method in C# and Visual Basic). It has no effect on other methods. To set the apartment state of threads you start in your code, use the Thread.SetApartmentState or Thread.TrySetApartmentState method before starting the thread.

> **Note**
>
> For an overview of COM threading models, see **Understanding and Using COM Threading Models**.

COM threading models only apply to applications that use COM interop. The COM threading model can be set to single-threaded apartment or multithreaded apartment. The application thread is only initialized for COM interop if the thread actually makes a call to a COM component. If COM interop is not used, then the thread is not initialized, and the MTAThreadAttribute attribute, if it is present, has no effect.

Starting with the .NET Framework version 2.0, the default threading model for COM interop depends on the language in which you are developing your application, as the following table shows.

Expand table

Language	COM apartment model
C#	Multithreaded apartment
C++	Multithreaded apartment
Visual Basic	Single-threaded apartment

To change these defaults, you use the MTAThreadAttribute attribute to set the threading model for the application, or call the Thread.SetApartmentState or Thread.TrySetApartmentState method before starting the thread to set the threading model for a particular thread. In C++, you can also use the /CLRTHREADATTRIBUTE (Set CLR Thread Attribute) linker option to specify the apartment model.

Some of the cases in which you want to use the MTAThreadAttribute attribute to explicitly set the threading model to multithreaded apartment include the following:

- You're developing a Visual Basic app that calls to a C# library that in turn relies on COM interop. Because the multithreaded apartment model is the default for C#, you should change your app's threading model to multithreaded by using the MTAThreadAttribute attribute.
- Your application makes calls to COM components that use the multithreaded apartment model.

Constructors

Expand table

MTAThreadAttribute()	Initializes a new instance of the MTAThreadAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute.

	(Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- STAThreadAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MTAThreadAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MTAThreadAttribute class.

C#

```
public MTAThreadAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- STAThreadAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastDelegate Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a multicast delegate; that is, a delegate that can have more than one element in its invocation list.

```
C#
public abstract class MulticastDelegate : Delegate
```

Inheritance Object → Delegate → MulticastDelegate

Examples

The following example defines a class, `StringContainer`, which includes a collection of strings. One of its members is the `CheckAndDisplayDelegate` delegate, which is used to display strings stored in a `StringContainer` object that satisfy particular criteria. The delegate takes a single string as a parameter and returns `void` (or, in Visual Basic, it's a `Sub` procedure). It also includes a method, `DisplayAllQualified`, that has a single parameter, a `CheckAndDisplayDelegate` delegate. This allows the method to be called and to display a set of strings that are filtered based on the methods that the delegate contains.

The example also defines a utility class, `StringExtensions`, that has two methods:

- `ConStart`, which displays strings that begin with a consonant.
- `VowelStart`, which displays strings that begin with a vowel.

Note that both methods include a single string parameter and return `void`. In other words, both methods can be assigned to the `CheckAndDisplayDelegate` delegate.

The `Test.Main` method is the application entry point. It instantiates a `StringContainer` object, populates it with strings, and creates two `CheckAndDisplayDelegate` delegates, `conStart` and `vowelStart`, that invoke a single method. It then calls the

Delegate.Combine method to create the `multipleDelegates` delegate, which initially contains the `ConStart` and `VowelStart` delegates. Note that when the `multipleDelegates` delegate is invoked, it displays all the strings in the collection in their original order. This is because each letter is passed separately to each delegate, and each letter meets the filtering criteria of only one of the two delegates. Finally, after calls to Delegate.Remove and Delegate.Combine, `multipleDelegates` contains two `conStart` delegates. When it is invoked, each string in the `StringContainer` object is displayed twice.

C#

```
using System;
using System.Collections.Generic;

class StringContainer
{
   // Define a delegate to handle string display.
   public delegate void CheckAndDisplayDelegate(string str);

   // A generic list object that holds the strings.
   private List<String> container = new List<String>();

   // A method that adds strings to the collection.
   public void AddString(string str)
   {
      container.Add(str);
   }

   // Iterate through the strings and invoke the method(s) that the delegate
points to.
   public void DisplayAllQualified(CheckAndDisplayDelegate displayDelegate)
   {
      foreach (var str in container) {
         displayDelegate(str);
      }
   }
 }

// This class defines some methods to display strings.
class StringExtensions
{
   // Display a string if it starts with a consonant.
   public static void ConStart(string str)
   {
      if (!
(str[0]=='a'||str[0]=='e'||str[0]=='i'||str[0]=='o'||str[0]=='u'))
          Console.WriteLine(str);
   }

   // Display a string if it starts with a vowel.
   public static void VowelStart(string str)
```

```
    {
        if ((str[0]=='a'||str[0]=='e'||str[0]=='i'||str[0]=='o'||str[0]=='u'))
            Console.WriteLine(str);
    }
}

// Demonstrate the use of delegates, including the Remove and
// Combine methods to create and modify delegate combinations.
class Test
{
    static public void Main()
    {
        // Declare the StringContainer class and add some strings
        StringContainer container = new StringContainer();
        container.AddString("This");
        container.AddString("is");
        container.AddString("a");
        container.AddString("multicast");
        container.AddString("delegate");
        container.AddString("example");

        // Create two delegates individually using different methods.
        StringContainer.CheckAndDisplayDelegate conStart =
StringExtensions.ConStart;
        StringContainer.CheckAndDisplayDelegate vowelStart =
StringExtensions.VowelStart;

        // Get the list of all delegates assigned to this MulticastDelegate
instance.
        Delegate[] delegateList = conStart.GetInvocationList();
        Console.WriteLine("conStart contains {0} delegate(s).",
delegateList.Length);
        delegateList = vowelStart.GetInvocationList();
        Console.WriteLine("vowelStart contains {0} delegate(s).\n",
delegateList.Length);

        // Determine whether the delegates are System.Multicast delegates.
        if (conStart is System.MulticastDelegate && vowelStart is
System.MulticastDelegate)
            Console.WriteLine("conStart and vowelStart are derived from
MulticastDelegate.\n");

        // Execute the two delegates.
        Console.WriteLine("Executing the conStart delegate:");
        container.DisplayAllQualified(conStart);
        Console.WriteLine();
        Console.WriteLine("Executing the vowelStart delegate:");
        container.DisplayAllQualified(vowelStart);
        Console.WriteLine();

        // Create a new MulticastDelegate and call Combine to add two
delegates.
        StringContainer.CheckAndDisplayDelegate multipleDelegates =
              (StringContainer.CheckAndDisplayDelegate)
Delegate.Combine(conStart, vowelStart);
```

```
      // How many delegates does multipleDelegates contain?
      delegateList = multipleDelegates.GetInvocationList();
      Console.WriteLine("\nmultipleDelegates contains {0} delegates.\n",
                        delegateList.Length);

      // Pass this multicast delegate to DisplayAllQualified.
      Console.WriteLine("Executing the multipleDelegate delegate.");
      container.DisplayAllQualified(multipleDelegates);

      // Call remove and combine to change the contained delegates.
      multipleDelegates = (StringContainer.CheckAndDisplayDelegate)
Delegate.Remove(multipleDelegates, vowelStart);
      multipleDelegates = (StringContainer.CheckAndDisplayDelegate)
Delegate.Combine(multipleDelegates, conStart);

      // Pass multipleDelegates to DisplayAllQualified again.
      Console.WriteLine("\nExecuting the multipleDelegate delegate with two
conStart delegates:");
      container.DisplayAllQualified(multipleDelegates);
   }
}
// The example displays the following output:
//    conStart contains 1 delegate(s).
//    vowelStart contains 1 delegate(s).
//
//    conStart and vowelStart are derived from MulticastDelegate.
//
//    Executing the conStart delegate:
//    This
//    multicast
//    delegate
//
//    Executing the vowelStart delegate:
//    is
//    a
//    example
//
//
//    multipleDelegates contains 2 delegates.
//
//    Executing the multipleDelegate delegate.
//    This
//    is
//    a
//    multicast
//    delegate
//    example
//
//    Executing the multipleDelegate delegate with two conStart delegates:
//    This
//    This
//    multicast
//    multicast
```

```
//    delegate
//    delegate
```

Remarks

MulticastDelegate is a special class. Compilers and other tools can derive from this class, but you cannot derive from it explicitly. The same is true of the Delegate class.

In addition to the methods that delegate types inherit from MulticastDelegate, the common language runtime provides two special methods: `BeginInvoke` and `EndInvoke`. For more information about these methods, see Calling Synchronous Methods Asynchronously.

A MulticastDelegate has a linked list of delegates, called an invocation list, consisting of one or more elements. When a multicast delegate is invoked, the delegates in the invocation list are called synchronously in the order in which they appear. If an error occurs during execution of the list then an exception is thrown.

Constructors

Expand table

MulticastDelegate(Object, String)	Initializes a new instance of the MulticastDelegate class.
MulticastDelegate(Type, String)	Initializes a new instance of the MulticastDelegate class.

Properties

Expand table

Method	Gets the method represented by the delegate. (Inherited from Delegate)
Target	Gets the class instance on which the current delegate invokes the instance method. (Inherited from Delegate)

Methods

Expand table

Clone()	Creates a shallow copy of the delegate. (Inherited from Delegate)
CombineImpl(Delegate)	Combines this Delegate with the specified Delegate to form a new delegate.
DynamicInvoke(Object[])	Dynamically invokes (late-bound) the method represented by the current delegate. (Inherited from Delegate)
DynamicInvokeImpl(Object[])	Dynamically invokes (late-bound) the method represented by the current delegate. (Inherited from Delegate)
Equals(Object)	Determines whether this multicast delegate and the specified object are equal.
GetHashCode()	Returns the hash code for this instance.
GetInvocationList()	Returns the invocation list of this multicast delegate, in invocation order.
GetMethodImpl()	Returns a method represented by the current MulticastDelegate.
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo object with all the data needed to serialize this instance.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
RemoveImpl(Delegate)	Removes an element from the invocation list of this MulticastDelegate that is equal to the specified delegate.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Operators

Expand table

Equality(MulticastDelegate, MulticastDelegate)	Determines whether two MulticastDelegate objects are equal.

Inequality(MulticastDelegate, MulticastDelegate)	Determines whether two MulticastDelegate objects are not equal.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastDelegate Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MulticastDelegate class.

Overloads

Expand table

MulticastDelegate(Object, String)	Initializes a new instance of the MulticastDelegate class.
MulticastDelegate(Type, String)	Initializes a new instance of the MulticastDelegate class.

MulticastDelegate(Object, String)

Initializes a new instance of the MulticastDelegate class.

C#

```
protected MulticastDelegate (object target, string method);
```

Parameters

target Object

The object on which `method` is defined.

method String

The name of the method for which a delegate is created.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

This constructor cannot be used in application code. To create a delegate by specifying the name of an instance method, use an overload of the Delegate.CreateDelegate method that specifies a method name and a target object. For example, the Delegate.CreateDelegate(Type, Object, String) method overload creates a delegate for an instance method with a specified name.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MulticastDelegate(Type, String)

Initializes a new instance of the MulticastDelegate class.

C#

```
protected MulticastDelegate (Type target, string method);
```

Parameters

`target` Type

The type of object on which `method` is defined.

`method` String

The name of the static method for which a delegate is created.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

This constructor cannot be used in application code. To create a delegate by specifying the name of a static method, use an overload of the Delegate.CreateDelegate method that specifies a method name but does not specify a target object. For example, the Delegate.CreateDelegate(Type, Type, String) method overload creates a static delegate for a method with a specified name.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastDelegate.CombineImpl(Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Combines this Delegate with the specified Delegate to form a new delegate.

C#

```
protected override sealed Delegate CombineImpl (Delegate? follow);
```

Parameters

`follow` Delegate

The delegate to combine with this delegate.

Returns

Delegate

A delegate that is the new root of the MulticastDelegate invocation list.

Exceptions

ArgumentException

`follow` does not have the same type as this instance.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastDelegate.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether this multicast delegate and the specified object are equal.

C#

```
public override sealed bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if `obj` and this instance have the same invocation lists; otherwise, `false`.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

Two delegates are equal if they are not `null` and are of exactly the same type, their invocation lists contain the same number of elements, and every element in the invocation list of the first delegate is equal to the corresponding element in the invocation list of the second delegate.

Two invocation list elements are equal if they invoke the same instance method on the same target instance, or they invoke the same static method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastDelegate.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override sealed int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastDelegate.GetInvocationList Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the invocation list of this multicast delegate, in invocation order.

C#

```
public override sealed Delegate[] GetInvocationList ();
```

Returns

Delegate[]

An array of delegates whose invocation lists collectively match the invocation list of this instance.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

The array contains a set of delegates, each with an invocation list of one entry. Invoking these delegates sequentially, in the order they appear in the array, produces the same results as invoking the current instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastDelegate.GetMethodImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a method represented by the current MulticastDelegate.

C#

```
protected override System.Reflection.MethodInfo GetMethodImpl ();
```

Returns

MethodInfo

A method represented by the current MulticastDelegate.

Remarks

If the MulticastDelegate represents multiple methods, only one method is returned.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastDelegate.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo object with all the data needed to serialize this instance.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

An object that holds all the data needed to serialize or deserialize this instance.

`context` StreamingContext

(Reserved) The location where serialized data is stored and retrieved.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

SerializationException

A serialization error occurred.

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

This method stores all the information in `info` necessary to serialize this instance.

The `context` parameter is reserved and is not currently used in the execution of this method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Object
- SerializationInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

MulticastDelegate.RemoveImpl(Delegate) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Removes an element from the invocation list of this MulticastDelegate that is equal to the specified delegate.

C#

```
protected override sealed Delegate? RemoveImpl (Delegate value);
```

Parameters

`value` Delegate

The delegate to search for in the invocation list.

Returns

Delegate

If `value` is found in the invocation list for this instance, then a new Delegate without `value` in its invocation list; otherwise, this instance with its original invocation list.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastDelegate. Equality(MulticastDelegate, MulticastDelegate) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two MulticastDelegate objects are equal.

```
C#

public static bool operator == (MulticastDelegate? d1, MulticastDelegate?
d2);
```

Parameters

d1 MulticastDelegate

The left operand.

d2 MulticastDelegate

The right operand.

Returns

Boolean

`true` if `d1` and `d2` have the same invocation lists; otherwise, `false`.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

Two delegates are equal if they are not `null` and are of exactly the same type, their invocation lists contain the same number of elements, and every element in the invocation list of the first delegate is equal to the corresponding element in the invocation list of the second delegate.

Two invocation list elements are equal if they invoke the same instance method on the same target instance, or they invoke the same static method.

The equivalent method for this operator is MulticastDelegate.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastDelegate.Inequality(MulticastDelegate, MulticastDelegate) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two MulticastDelegate objects are not equal.

```
C#

public static bool operator != (MulticastDelegate? d1, MulticastDelegate?
d2);
```

Parameters

d1 MulticastDelegate

The left operand.

d2 MulticastDelegate

The right operand.

Returns

Boolean

`true` if `d1` and `d2` do not have the same invocation lists; otherwise, `false`.

Exceptions

MemberAccessException

Cannot create an instance of an abstract class, or this member was invoked with a late-binding mechanism.

Remarks

Two delegates are equal if they are not `null` and are of exactly the same type, their invocation lists contain the same number of elements, and every element in the invocation list of the first delegate is equal to the corresponding element in the invocation list of the second delegate.

Two invocation list elements are equal if they invoke the same instance method on the same target instance, or they invoke the same static method.

The equivalent method for this operator is MulticastDelegate.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

MulticastNotSupportedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to combine two delegates based on the Delegate type instead of the MulticastDelegate type. This class cannot be inherited.

C#

```
public sealed class MulticastNotSupportedException : SystemException
```

Inheritance Object → Exception → SystemException → MulticastNotSupportedException

Remarks

Only delegates based on the MulticastDelegate type can be combined. Two multicast delegates can be combined using the CombineImpl method.

MulticastNotSupportedException uses the HRESULT COR_E_MULTICASTNOTSUPPORTED, which has the value 0x80131514.

For a list of initial property values for an instance of MulticastNotSupportedException, see the MulticastNotSupportedException constructors.

Constructors

Expand table

MulticastNotSupportedException()	Initializes a new instance of the MulticastNotSupportedException class.
MulticastNotSupportedException(String)	Initializes a new instance of the MulticastNotSupportedException class with a specified error message.

MulticastNotSupportedException(String, Exception)	Initializes a new instance of the MulticastNotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
InnerException	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)

GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- MulticastDelegate
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

MulticastNotSupportedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the MulticastNotSupportedException class.

Overloads

Expand table

MulticastNotSupportedException()	Initializes a new instance of the MulticastNotSupportedException class.
MulticastNotSupportedException(String)	Initializes a new instance of the MulticastNotSupportedException class with a specified error message.
MulticastNotSupportedException(String, Exception)	Initializes a new instance of the MulticastNotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

MulticastNotSupportedException()

Initializes a new instance of the MulticastNotSupportedException class.

C#

```
public MulticastNotSupportedException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "This operation cannot be

performed with the specified delegates." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MulticastNotSupportedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MulticastNotSupportedException(String)

Initializes a new instance of the MulticastNotSupportedException class with a specified error message.

C#

```
public MulticastNotSupportedException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of MulticastNotSupportedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

MulticastNotSupportedException(String, Exception)

Initializes a new instance of the MulticastNotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public MulticastNotSupportedException (string? message, Exception?
inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of MulticastNotSupportedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

NetPipeStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A parser based on the NetPipe scheme for the "Indigo" system.

C#

```
public class NetPipeStyleUriParser : UriParser
```

Inheritance Object → UriParser → NetPipeStyleUriParser

Remarks

This parser is for future use with the "Indigo" system.

Constructors

Expand table

NetPipeStyleUriParser()	Create a parser based on the NetPipe scheme for the "Indigo" system.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)

GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

NetPipeStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Create a parser based on the NetPipe scheme for the "Indigo" system.

C#

```
public NetPipeStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NetTcpStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A parser based on the NetTcp scheme for the "Indigo" system.

C#

```
public class NetTcpStyleUriParser : UriParser
```

Inheritance Object → UriParser → NetTcpStyleUriParser

Remarks

This parser is for future use with the "Indigo" system.

Constructors

Expand table

NetTcpStyleUriParser()	Create a parser based on the NetTcp scheme for the "Indigo" system.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)

GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

NetTcpStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Create a parser based on the NetTcp scheme for the "Indigo" system.

C#

```
public NetTcpStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NewsStyleUriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A customizable parser based on the news scheme using the Network News Transfer Protocol (NNTP).

C#

```
public class NewsStyleUriParser : UriParser
```

Inheritance Object → UriParser → NewsStyleUriParser

Constructors

Expand table

NewsStyleUriParser()	Create a customizable parser based on the news scheme using the Network News Transfer Protocol (NNTP).

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI. (Inherited from UriParser)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)

InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI. (Inherited from UriParser)
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`. (Inherited from UriParser)
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed. (Inherited from UriParser)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance. (Inherited from UriParser)
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered. (Inherited from UriParser)
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI. (Inherited from UriParser)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

NewsStyleUriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Create a customizable parser based on the news scheme using the Network News Transfer Protocol (NNTP).

C#

```
public NewsStyleUriParser ();
```

Remarks

Use this constructor as an argument to UriParser.Register(UriParser, String, Int32)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NonSerializedAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that a field of a serializable class should not be serialized. This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.Field, Inherited=false)]
public sealed class NonSerializedAttribute : Attribute
```

Inheritance Object → Attribute → NonSerializedAttribute

Attributes AttributeUsageAttribute

Examples

The following example demonstrates SOAP serialization of an object marked with the SerializableAttribute attribute, and the behavior of a field marked with the NonSerializedAttribute in the serialized object.

> **Note**
>
> The code uses the **SoapFormatter** class to serialize the object. The class is found in the system.runtime.serialization.formatters.soap.dll file, which is not loaded by default into a project. To run the code, you must add a reference to the DLL to your project.

C#

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

public class Test {
```

```
    public static void Main()  {

      // Creates a new TestSimpleObject object.
      TestSimpleObject obj = new TestSimpleObject();

      Console.WriteLine("Before serialization the object contains: ");
      obj.Print();

      // Opens a file and serializes the object into it in binary format.
      Stream stream = File.Open("data.xml", FileMode.Create);
      SoapFormatter formatter = new SoapFormatter();

      formatter.Serialize(stream, obj);
      stream.Close();

      // Empties obj.
      obj = null;

      // Opens file "data.xml" and deserializes the object from it.
      stream = File.Open("data.xml", FileMode.Open);
      formatter = new SoapFormatter();

      obj = (TestSimpleObject)formatter.Deserialize(stream);
      stream.Close();

      Console.WriteLine("");
      Console.WriteLine("After deserialization the object contains: ");
      obj.Print();
    }
}

// A test object that needs to be serialized.
[Serializable()]
public class TestSimpleObject  {

    public int member1;
    public string member2;
    public string member3;
    public double member4;

    // A field that is not serialized.
    [NonSerialized()] public string member5;

    public TestSimpleObject() {

        member1 = 11;
        member2 = "hello";
        member3 = "hello";
        member4 = 3.14159265;
        member5 = "hello world!";
    }

    public void Print() {

        Console.WriteLine("member1 = '{0}'", member1);
```

```
        Console.WriteLine("member2 = '{0}'", member2);
        Console.WriteLine("member3 = '{0}'", member3);
        Console.WriteLine("member4 = '{0}'", member4);
        Console.WriteLine("member5 = '{0}'", member5);
    }
}
```

Remarks

When using the BinaryFormatter or SoapFormatter classes to serialize an object, use the NonSerializedAttribute attribute to prevent a field from being serialized. For example, you can use this attribute to prevent the serialization of sensitive data.

The target objects for the NonSerializedAttribute attribute are public and private fields of a serializable class. By default, classes are not serializable unless they are marked with SerializableAttribute. During the serialization process all the public and private fields of a class are serialized by default. Fields marked with NonSerializedAttribute are excluded during serialization. If you are using the XmlSerializer class to serialize an object, use the XmlIgnoreAttribute class to get the same functionality. Alternatively, implement the ISerializable interface to explicitly control the serialization process. Note that classes that implement ISerializable must still be marked with SerializableAttribute.

To apply the NonSerializedAttribute class to an event, set the attribute location to field, as shown in the following C# code.

C#

```
[field:NonSerializedAttribute()]
public event ChangedEventHandler Changed;
```

If a field is not serialized, but it still requires a default value that must be supplied after deserialization, you can create a method that supplies the field with a value, then apply the OnDeserializedAttribute to the method.

For more information about using attributes, see Attributes.

> **Note**
>
> This attribute does not apply to JSON serialization using **System.Text.Json**.

Constructors

Expand table

NonSerializedAttribute()	Initializes a new instance of the NonSerializedAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializableAttribute
- Attribute
- ISerializable
- Extending Metadata Using Attributes
- XML and SOAP Serialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NonSerializedAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the NonSerializedAttribute class.

C#

```
public NonSerializedAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotFiniteNumberException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a floating-point value is positive infinity, negative infinity, or Not-a-Number (NaN).

C#

```
public class NotFiniteNumberException : ArithmeticException
```

Inheritance Object → Exception → SystemException → ArithmeticException → NotFiniteNumberException

Remarks

NotFiniteNumberException is available for programming languages that do not support the concepts of infinity and Not-a-Number in floating-point operations.

NotFiniteNumberException uses the HRESULT COR_E_NOTFINITENUMBER, which has the value 0x80131528.

For a list of initial property values for an instance of NotFiniteNumberException, see the NotFiniteNumberException constructors.

Constructors

Expand table

NotFiniteNumberException()	Initializes a new instance of the NotFiniteNumberException class.
NotFiniteNumber Exception(Double)	Initializes a new instance of the NotFiniteNumberException class with the invalid number.
NotFiniteNumber Exception(SerializationInfo,	**Obsolete.**

StreamingContext)	Initializes a new instance of the NotFiniteNumberException class with serialized data.
NotFiniteNumberException(String)	Initializes a new instance of the NotFiniteNumberException class with a specified error message.
NotFiniteNumberException(String, Double)	Initializes a new instance of the NotFiniteNumberException class with a specified error message and the invalid number.
NotFiniteNumberException(String, Double, Exception)	Initializes a new instance of the NotFiniteNumberException class with a specified error message, the invalid number, and a reference to the inner exception that is root cause of this exception.
NotFiniteNumberException(String, Exception)	Initializes a new instance of the NotFiniteNumberException class with a specified error message and a reference to the inner exception that is root cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
InnerException	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Offending Number	Gets the invalid number that is a positive infinity, a negative infinity, or Not-a-Number (NaN).
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception.

	(Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the invalid number and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- Double
- Single
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotFiniteNumberException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the NotFiniteNumberException class.

Overloads

Expand table

NotFiniteNumberException()	Initializes a new instance of the NotFiniteNumberException class.
NotFiniteNumberException(Double)	Initializes a new instance of the NotFiniteNumberException class with the invalid number.
NotFiniteNumberException(String)	Initializes a new instance of the NotFiniteNumberException class with a specified error message.
NotFiniteNumberException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NotFiniteNumberException class with serialized data.
NotFiniteNumberException(String, Double)	Initializes a new instance of the NotFiniteNumberException class with a specified error message and the invalid number.
NotFiniteNumberException(String, Exception)	Initializes a new instance of the NotFiniteNumberException class with a specified error message and a reference to the inner exception that is root cause of this exception.
NotFiniteNumberException(String, Double, Exception)	Initializes a new instance of the NotFiniteNumberException class with a specified error message, the invalid number, and a

	reference to the inner exception that is root cause of this exception.

NotFiniteNumberException()

Initializes a new instance of the NotFiniteNumberException class.

```
C#

public NotFiniteNumberException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The number encountered was not a finite quantity." This message takes into account the current system culture.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(Double)

Initializes a new instance of the NotFiniteNumberException class with the invalid number.

```
C#

public NotFiniteNumberException (double offendingNumber);
```

Parameters

`offendingNumber` Double

The value of the argument that caused the exception.

Remarks

The `offendingNumber` parameter must be both a Double and an invalid number.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
OffendingNumber	The invalid number.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(String)

Initializes a new instance of the NotFiniteNumberException class with a specified error message.

```
C#

public NotFiniteNumberException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

Ensure that the `message` string is understandable to the user and that it is localized for the current culture.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the NotFiniteNumberException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected NotFiniteNumberException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(String, Double)

Initializes a new instance of the NotFiniteNumberException class with a specified error message and the invalid number.

C#

```
public NotFiniteNumberException (string? message, double
offendingNumber);
```

Parameters

`message` String

The message that describes the error.

`offendingNumber` Double

The value of the argument that caused the exception.

Remarks

Ensure that the `message` string is understandable to the user and that it is localized for the current culture.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Property	Value
OffendingNumber	The invalid number.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(String, Exception)

Initializes a new instance of the NotFiniteNumberException class with a specified error message and a reference to the inner exception that is root cause of this exception.

C#

```
public NotFiniteNumberException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotFiniteNumberException(String, Double, Exception)

Initializes a new instance of the NotFiniteNumberException class with a specified error message, the invalid number, and a reference to the inner exception that is root cause of this exception.

C#

```
public NotFiniteNumberException (string? message, double offendingNumber,
Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`offendingNumber` Double

The value of the argument that caused the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of NotFiniteNumberException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.
OffendingNumber	The invalid number.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotFiniteNumberException.OffendingNumber Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the invalid number that is a positive infinity, a negative infinity, or Not-a-Number (NaN).

C#

```
public double OffendingNumber { get; }
```

Property Value

Double

The invalid number.

Remarks

The OffendingNumber property returns the same value as was passed into the constructor, or zero if no value was supplied. Override the OffendingNumber property to customize the content or format of the invalid value.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Double

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotFiniteNumberException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the invalid number and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is `null`.

Remarks

GetObjectData sets a SerializationInfo with all the exception object data targeted for serialization. During deserialization, the exception object is reconstituted from the SerializationInfo transmitted over the stream.

For more information, see SerializationInfo.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- StreamingContext

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotImplementedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a requested method or operation is not implemented.

C#

```
public class NotImplementedException : SystemException
```

Inheritance Object → Exception → SystemException → NotImplementedException

Examples

The following example throws this exception for a method that has not been developed.

C#

```
static void Main(string[] args)
{
    try
    {
        FutureFeature();
    }
    catch (NotImplementedException notImp)
    {
        Console.WriteLine(notImp.Message);
    }
}

static void FutureFeature()
{
    // Not developed yet.
    throw new NotImplementedException();
}
```

Remarks

For more information about this API, see Supplemental API remarks for NotImplementedException.

Constructors

Expand table

NotImplementedException()	Initializes a new instance of the NotImplementedException class with default properties.
NotImplemented Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NotImplementedException class with serialized data.
NotImplementedException(String)	Initializes a new instance of the NotImplementedException class with a specified error message.
NotImplementedException(String, Exception)	Initializes a new instance of the NotImplementedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack.

	(Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- NotSupportedException

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotImplementedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the NotImplementedException class.

Overloads

Expand table

NotImplementedException()	Initializes a new instance of the NotImplementedException class with default properties.
NotImplementedException(String)	Initializes a new instance of the NotImplementedException class with a specified error message.
NotImplementedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NotImplementedException class with serialized data.
NotImplementedException(String, Exception)	Initializes a new instance of the NotImplementedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

NotImplementedException()

Initializes a new instance of the NotImplementedException class with default properties.

C#

```
public NotImplementedException ();
```

Remarks

The following table shows the initial property values for an instance of NotImplementedException.

Expand table

Property	Value
InnerException	`null`.
Message	The empty string ("").

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NotImplementedException(String)

Initializes a new instance of the NotImplementedException class with a specified error message.

C#

```
public NotImplementedException (string? message);
```

Parameters

`message` String

The error message that explains the reason for the exception.

Remarks

The following table shows the initial property values for an instance of NotImplementedException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NotImplementedException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the NotImplementedException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected NotImplementedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotImplementedException(String, Exception)

Initializes a new instance of the NotImplementedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public NotImplementedException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of NotImplementedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

NotSupportedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an invoked method is not supported, or when there is an attempt to read, seek, or write to a stream that does not support the invoked functionality.

C#

```
public class NotSupportedException : SystemException
```

Inheritance Object → Exception → SystemException → NotSupportedException

Derived System.PlatformNotSupportedException

Remarks

For more information about this API, see Supplemental API remarks for NotSupportedException.

Constructors

Expand table

NotSupportedException()	Initializes a new instance of the NotSupportedException class, setting the Message property of the new instance to a system-supplied message that describes the error. This message takes into account the current system culture.
NotSupportedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NotSupportedException class with serialized data.
NotSupportedException(String)	Initializes a new instance of the NotSupportedException class with a specified error message.

NotSupportedException(String, Exception)	Initializes a new instance of the NotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)

GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- FileStream

- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NotSupportedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the NotSupportedException class.

Overloads

Expand table

NotSupportedException()	Initializes a new instance of the NotSupportedException class, setting the Message property of the new instance to a system-supplied message that describes the error. This message takes into account the current system culture.
NotSupportedException(String)	Initializes a new instance of the NotSupportedException class with a specified error message.
NotSupportedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NotSupportedException class with serialized data.
NotSupportedException(String, Exception)	Initializes a new instance of the NotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

NotSupportedException()

Initializes a new instance of the NotSupportedException class, setting the Message property of the new instance to a system-supplied message that describes the error. This message takes into account the current system culture.

```
C#
```

```
public NotSupportedException ();
```

Remarks

The following table shows the initial property values for an instance of NotSupportedException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NotSupportedException(String)

Initializes a new instance of the NotSupportedException class with a specified error message.

C#

```
public NotSupportedException (string? message);
```

Parameters

`message` String

A String that describes the error. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of NotSupportedException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NotSupportedException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the NotSupportedException class with serialized data.

```
C#

[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected NotSupportedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NotSupportedException(String, Exception)

Initializes a new instance of the NotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public NotSupportedException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of NotSupportedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and throwing exceptions in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Supports a value type that can be assigned `null`. This class cannot be inherited.

C#

```
public static class Nullable
```

Inheritance Object → Nullable

Remarks

For more information about this API, see Supplemental API remarks for Nullable.

Methods

Expand table

Compare<T>(Nullable<T>, Nullable<T>)	Compares the relative values of two Nullable<T> objects.
Equals<T>(Nullable<T>, Nullable<T>)	Indicates whether two specified Nullable<T> objects are equal.
GetUnderlyingType(Type)	Returns the underlying type argument of the specified nullable type.
GetValueRefOrDefaultRef<T>(Nullable<T>)	Retrieves a readonly reference to the location in the Nullable<T> instance where the value is stored.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Nullable<T>
- Nullable Types (C# Programming Guide)
- Nullable Value Types (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable.Compare<T>(Nullable<T>, Nullable<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the relative values of two Nullable<T> objects.

C#

```
public static int Compare<T> (T? n1, T? n2) where T : struct;
```

Type Parameters

T

The underlying value type of the `n1` and `n2` parameters.

Parameters

n1 Nullable<T>

A Nullable<T> object.

n2 Nullable<T>

A Nullable<T> object.

Returns

Int32

An integer that indicates the relative values of the `n1` and `n2` parameters.

Expand table

Return Value	Description
Less than zero	The HasValue property for `n1` is `false`, and the HasValue property for `n2` is `true`, or the HasValue properties for `n1` and `n2` are `true`, and the value of the Value property for `n1` is less than the value of the Value property for `n2`.
Zero	The HasValue properties for `n1` and `n2` are `false`, or the HasValue properties for `n1` and `n2` are `true`, and the value of the Value property for `n1` is equal to the value of the Value property for `n2`.
Greater than zero	The HasValue property for `n1` is `true`, and the HasValue property for `n2` is `false`, or the HasValue properties for `n1` and `n2` are `true`, and the value of the Value property for `n1` is greater than the value of the Value property for `n2`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable.Equals<T>(Nullable<T>, Nullable<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two specified Nullable<T> objects are equal.

C#

```
public static bool Equals<T> (T? n1, T? n2) where T : struct;
```

Type Parameters

T

The underlying value type of the `n1` and `n2` parameters.

Parameters

n1 Nullable<T>

A Nullable<T> object.

n2 Nullable<T>

A Nullable<T> object.

Returns

Boolean

`true` if the `n1` parameter is equal to the `n2` parameter; otherwise, `false`.

The return value depends on the HasValue and Value properties of the two parameters that are compared.

Expand table

Return Value	Description
`true`	The HasValue properties for `n1` and `n2` are `false`, or the HasValue properties for `n1` and `n2` are `true`, and the Value properties of the parameters are equal.
`false`	The HasValue property is `true` for one parameter and `false` for the other parameter, or the HasValue properties for `n1` and `n2` are `true`, and the Value properties of the parameters are unequal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable.GetUnderlyingType(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the underlying type argument of the specified nullable type.

```
C#
public static Type? GetUnderlyingType (Type nullableType);
```

Parameters

`nullableType` Type

A Type object that describes a closed generic nullable type.

Returns

Type

The type argument of the `nullableType` parameter, if the `nullableType` parameter is a closed generic nullable type; otherwise, `null`.

Exceptions

ArgumentNullException

`nullableType` is `null`.

Examples

The following code example defines a method whose return value is of type Nullable<T> of Int32. The code example uses the GetUnderlyingType method to display the type argument of the return value.

```
C#
```

```
// This code example demonstrates the
// Nullable.GetUnderlyingType() method.

using System;
using System.Reflection;

class Sample
{
// Declare a type named Example.
// The MyMethod member of Example returns a Nullable of Int32.

    public class Example
    {
        public int? MyMethod()
        {
        return 0;
        }
    }

/*
   Use reflection to obtain a Type object for the Example type.
   Use the Type object to obtain a MethodInfo object for the MyMethod
method.
   Use the MethodInfo object to obtain the type of the return value of
     MyMethod, which is Nullable of Int32.
   Use the GetUnderlyingType method to obtain the type argument of the
     return value type, which is Int32.
*/
    public static void Main()
    {
        Type t = typeof(Example);
        MethodInfo mi = t.GetMethod("MyMethod");
        Type retval = mi.ReturnType;
        Console.WriteLine("Return value type ... {0}", retval);
        Type answer = Nullable.GetUnderlyingType(retval);
        Console.WriteLine("Underlying type ..... {0}", answer);
    }
}
/*
This code example produces the following results:

Return value type ... System.Nullable`1[System.Int32]
Underlying type ..... System.Int32

*/
```

Remarks

A generic type definition is a type declaration, such as Nullable<T>, that contains a type parameter list, and the type parameter list declares one or more type parameters. A

closed generic type is a type declaration where a particular type is specified for a type parameter.

For example, if the `nullableType` parameter is the type of `Nullable<Int32>` in C# (`Nullable(Of Int32)` in Visual Basic), the return value is the type of Int32 (that is, the type argument of the closed generic type).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable.GetValueRefOrDefaultRef<T> (Nullable<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves a readonly reference to the location in the Nullable<T> instance where the value is stored.

C#

```
public static ref readonly T GetValueRefOrDefaultRef<T> (ref T? nullable)
where T : struct;
```

Type Parameters

T

The underlying value type of the Nullable<T> generic type.

Parameters

nullable Nullable<T>

The readonly reference to the input Nullable<T> value.

Returns

T

A readonly reference to the location where the instance's `T` value is stored. If the instance's HasValue is `false`, the current value at that location may be the default value.

Remarks

As the returned readonly reference refers to data that is stored in the input `nullable` value, this method should only ever be called when the input reference points to a value with an actual location and not an "rvalue" (an expression that may appear on the right

side but not left side of an assignment). That is, if this API is called and the input reference points to a value that is produced by the compiler as a defensive copy or a temporary copy, the behavior might not match the desired one.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value type that can be assigned `null`.

C#

```
public struct Nullable<T> where T : struct
```

Type Parameters

T

The underlying value type of the Nullable<T> generic type.

Inheritance Object → ValueType → Nullable<T>

Examples

The following code example defines three rows of a table in the Microsoft Pubs sample database. The table contains two columns that are not nullable and two columns that are nullable.

C#

```
using System;

class Sample
{
    // Define the "titleAuthor" table of the Microsoft "pubs" database.
    public struct titleAuthor
    {
      // Author ID; format ###-##-####
      public string au_id;
      // Title ID; format AA####
      public string title_id;
      // Author ORD is nullable.
      public short? au_ord;
      // Royalty Percent is nullable.
```

```
        public int? royaltyper;
      }

      public static void Main()
      {
        // Declare and initialize the titleAuthor array.
        titleAuthor[] ta = new titleAuthor[3];
        ta[0].au_id = "712-32-1176";
        ta[0].title_id = "PS3333";
        ta[0].au_ord = 1;
        ta[0].royaltyper = 100;

        ta[1].au_id = "213-46-8915";
        ta[1].title_id = "BU1032";
        ta[1].au_ord = null;
        ta[1].royaltyper = null;

        ta[2].au_id = "672-71-3249";
        ta[2].title_id = "TC7777";
        ta[2].au_ord = null;
        ta[2].royaltyper = 40;

        // Display the values of the titleAuthor array elements, and
        // display a legend.
        Display("Title Authors Table", ta);
        Console.WriteLine("Legend:");
        Console.WriteLine("An Author ORD of -1 means no value is defined.");
        Console.WriteLine("A Royalty % of 0 means no value is defined.");
      }

      // Display the values of the titleAuthor array elements.
      public static void Display(string dspTitle,
                                 titleAuthor[] dspAllTitleAuthors)
      {
        Console.WriteLine("*** {0} ***", dspTitle);
        foreach (titleAuthor dspTA in dspAllTitleAuthors) {
           Console.WriteLine("Author ID ... {0}", dspTA.au_id);
           Console.WriteLine("Title ID .... {0}", dspTA.title_id);
           Console.WriteLine("Author ORD .. {0}", dspTA.au_ord ?? -1);
           Console.WriteLine("Royalty % ... {0}", dspTA.royaltyper ?? 0);
           Console.WriteLine();
        }
      }
  }
  // The example displays the following output:
  //     *** Title Authors Table ***
  //     Author ID ... 712-32-1176
  //     Title ID .... PS3333
  //     Author ORD .. 1
  //     Royalty % ... 100
  //
  //     Author ID ... 213-46-8915
  //     Title ID .... BU1032
  //     Author ORD .. -1
  //     Royalty % ... 0
```

```
//
//      Author ID ... 672-71-3249
//      Title ID .... TC7777
//      Author ORD .. -1
//      Royalty % ... 40
//
//      Legend:
//      An Author ORD of -1 means no value is defined.
//      A Royalty % of 0 means no value is defined.
```

Remarks

For more information about this API, see Supplemental API remarks for Nullable<T>.

Constructors

Expand table

Nullable<T>(T)	Initializes a new instance of the Nullable<T> structure to the specified value.

Properties

Expand table

Has Value	Gets a value indicating whether the current Nullable<T> object has a valid value of its underlying type.
Value	Gets the value of the current Nullable<T> object if it has been assigned a valid underlying value.

Methods

Expand table

Equals(Object)	Indicates whether the current Nullable<T> object is equal to a specified object.
GetHashCode()	Retrieves the hash code of the object returned by the Value property.
GetValue OrDefault()	Retrieves the value of the current Nullable<T> object, or the default value of the underlying type.

GetValueOrDefault(T)	Retrieves the value of the current Nullable<T> object, or the specified default value.
ToString()	Returns the text representation of the value of the current Nullable<T> object.

Operators

Expand table

Explicit(Nullable<T> to T)	Defines an explicit conversion of a Nullable<T> instance to its underlying value.
Implicit(T to Nullable<T>)	Creates a new Nullable<T> object initialized to a specified value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Nullable
- Nullable Types (C# Programming Guide)
- Nullable Value Types (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Nullable<T>(T) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Nullable<T> structure to the specified value.

C#

```
public Nullable (T value);
```

Parameters

`value` T

A value type.

Remarks

The Nullable<T> constructor initializes the HasValue property of the new Nullable<T> object to `true`, and the Value property to the value of the `value` parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Nullable<T>.HasValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Nullable<T> object has a valid value of its underlying type.

```
C#

public bool HasValue { get; }
```

Property Value

Boolean

`true` if the current Nullable<T> object has a value; `false` if the current Nullable<T> object has no value.

Examples

The following example uses the HasValue property of a `Nullable<int>` (`Nullable(Of Integer)` in Visual Basic) object to determine whether it should display the object's Value property or its GetValueOrDefault property.

```
C#

using System;

public class Example
{
   public static void Main()
   {
      Nullable<int> n1 = new Nullable<int>(10);
      Nullable<int> n2 = null;
      Nullable<int> n3 = new Nullable<int>(20);
      n3 = null;
      Nullable<int>[] items = { n1, n2, n3 };

      foreach (var item in items) {
```

```
            Console.WriteLine("Has a value: {0}", item.HasValue);
            if (item.HasValue) {
               Console.WriteLine("Type: {0}", item.GetType().Name);
               Console.WriteLine("Value: {0}", item.Value);
            }
            else {
               Console.WriteLine("Null: {0}", item == null);
               Console.WriteLine("Default Value: {0}",
item.GetValueOrDefault());
            }
            Console.WriteLine();
         }
      }
}
// The example displays the following output:
//       Has a value: True
//       Type: Int32
//       Value: 10
//
//       Has a value: False
//       Null: True
//       Default Value: 0
//
//       Has a value: False
//       Null: True
//       Default Value: 0
```

Remarks

If the HasValue property is `true`, the value of the current Nullable<T> object can be accessed with the Value property. Otherwise, attempting to access its value throws an InvalidOperationException exception.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Nullable<T>.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Nullable<T> object if it has been assigned a valid underlying value.

```
C#

public T Value { get; }
```

Property Value

T

The value of the current Nullable<T> object if the HasValue property is `true`. An exception is thrown if the HasValue property is `false`.

Exceptions

InvalidOperationException

The HasValue property is `false`.

Examples

The following example uses the HasValue property of a `Nullable<int>` (`Nullable(Of Integer)` in Visual Basic) object to determine whether it should display the object's Value property or its GetValueOrDefault property.

```
C#

using System;

public class Example
{
   public static void Main()
   {
      Nullable<int> n1 = new Nullable<int>(10);
```

```
        Nullable<int> n2 = null;
        Nullable<int> n3 = new Nullable<int>(20);
        n3 = null;
        Nullable<int>[] items = { n1, n2, n3 };

        foreach (var item in items) {
           Console.WriteLine("Has a value: {0}", item.HasValue);
           if (item.HasValue) {
              Console.WriteLine("Type: {0}", item.GetType().Name);
              Console.WriteLine("Value: {0}", item.Value);
           }
           else {
              Console.WriteLine("Null: {0}", item == null);
              Console.WriteLine("Default Value: {0}",
item.GetValueOrDefault());
           }
           Console.WriteLine();
        }
    }
}
// The example displays the following output:
//       Has a value: True
//       Type: Int32
//       Value: 10
//
//       Has a value: False
//       Null: True
//       Default Value: 0
//
//       Has a value: False
//       Null: True
//       Default Value: 0
```

Remarks

If a value of type `T` has not been assigned to the Nullable<T> object, you can compare it to `null` and retrieve its HasValue property, but you cannot access its Value property or call its other members.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Nullable<T>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current Nullable<T> object is equal to a specified object.

C#

```
public override bool Equals (object? other);
```

Parameters

`other` Object

An object.

Returns

Boolean

`true` if the `other` parameter is equal to the current Nullable<T> object; otherwise, `false`.

This table describes how equality is defined for the compared values:

Expand table

Return Value	Description
`true`	The HasValue property is `false`, and the `other` parameter is `null` (that is, two null values are equal by definition), OR the HasValue property is `true`, and the value returned by the Value property is equal to the `other` parameter.
`false`	The HasValue property for the current Nullable<T> structure is `true`, and the `other` parameter is `null`, OR the HasValue property for the current Nullable<T> structure is `false`, and the `other` parameter is not `null`, OR the HasValue property for the current Nullable<T> structure is `true`, and the value returned by the Value property is not equal to the `other` parameter.

Examples

The following code example determines whether an object and a Nullable<T> object are equal to the current Nullable<T> object.

C#

```
// This code example demonstrates the Nullable<T>.Equals
// methods.

using System;

class Sample
{
    public static void Main()
    {
    int? nullInt1 = 100;
    int? nullInt2 = 200;
    object myObj;

// Determine if two nullable of System.Int32 values are equal.
// The nullable objects have different values.
    Console.Write("1) nullInt1 and nullInt2 ");
    if (nullInt1.Equals(nullInt2))
        Console.Write("are");
    else
        Console.Write("are not");
    Console.WriteLine(" equal.");

// Determine if a nullable of System.Int32 and an object
// are equal. The object contains the boxed value of the
// nullable object.

    myObj = (object)nullInt1;
    Console.Write("2) nullInt1 and myObj ");
    if (nullInt1.Equals(myObj))
        Console.Write("are");
    else
        Console.Write("are not");
    Console.WriteLine(" equal.");
    }
}

/*
This code example produces the following results:

1) nullInt1 and nullInt2 are not equal.
2) nullInt1 and myObj are equal.

*/
```

Remarks

If the HasValue property of the current Nullable<T> structure is `true` and the `other` argument is not `null`, equality is determined by passing the `other` parameter to the `Equals` method of the underlying value of the current Nullable<T> structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the hash code of the object returned by the Value property.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code of the object returned by the Value property if the HasValue property is `true`, or zero if the HasValue property is `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Nullable<T>.GetValueOrDefault Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the value of the current Nullable<T> object, or a default value.

Overloads

Expand table

GetValueOrDefault()	Retrieves the value of the current Nullable<T> object, or the default value of the underlying type.
GetValueOrDefault(T)	Retrieves the value of the current Nullable<T> object, or the specified default value.

Examples

The following code example retrieves the value of a Nullable<T> object if that value is defined; otherwise, it retrieves the default value or a specific default value.

C#

```
// This code example demonstrates the
// Nullable<T>.GetValueOrDefault methods.

using System;

class Sample
{
    public static void Main()
    {
    float? mySingle = 12.34f;
    float? yourSingle = -1.0f;

   Console.WriteLine("*** Display a value or the default value ***\n");
// Display the values of mySingle and yourSingle.

    Display("A1", mySingle, yourSingle);
```

```
// Assign the value of mySingle to yourSingle, then display the values
// of mySingle and yourSingle. The yourSingle variable is assigned the
// value 12.34 because mySingle has a value.

    yourSingle = mySingle.GetValueOrDefault();
    Display("A2", mySingle, yourSingle);

// Assign null (Nothing in Visual Basic) to mySingle, which means no value
is
// defined for mySingle. Then assign the value of mySingle to yourSingle and
// display the values of both variables. The default value of all binary
zeroes
// is assigned to yourSingle because mySingle has no value.

    mySingle = null;
    yourSingle = mySingle.GetValueOrDefault();
    Display("A3", mySingle, yourSingle);

// Reassign the original values of mySingle and yourSingle.
    mySingle = 12.34f;
    yourSingle = -1.0f;

    Console.Write("\n*** Display a value or the ");
    Console.WriteLine("specified default value ***\n");

// Display the values of mySingle and yourSingle.
    Display("B1", mySingle, yourSingle);

// Assign the value of mySingle to yourSingle, then display the values
// of mySingle and yourSingle. The yourSingle variable is assigned the
// value 12.34 because mySingle has a value.

    yourSingle = mySingle.GetValueOrDefault(-222.22f);
    Display("B2", mySingle, yourSingle);

// Assign null (Nothing in Visual Basic) to mySingle, which means no value
is
// defined for mySingle. Then assign the value of mySingle to yourSingle and
// display the values of both variables. The specified default value of
-333.33
// is assigned to yourSingle because mySingle has no value.

    mySingle = null;
    yourSingle = mySingle.GetValueOrDefault(-333.33f);
    Display("B3", mySingle, yourSingle);
    }

// Display the values of two nullable of System.Single structures.
// The Console.WriteLine method automatically calls the ToString methods of
// each input argument to display its values. If no value is defined for a
// nullable type, the ToString method for that argument returns the empty
// string ("").
    public static void Display(string title, float? dspMySingle, float?
dspYourSingle)
    {
```

```
    Console.WriteLine("{0}) mySingle = [{1}], yourSingle = [{2}]",
                      title, dspMySingle, dspYourSingle);
    }
}

/*
This code example produces the following results:

A1) mySingle = [12.34], yourSingle = [-1]
A2) mySingle = [12.34], yourSingle = [12.34]
A3) mySingle = [], yourSingle = [0]

*** Display a value or the specified default value ***

B1) mySingle = [12.34], yourSingle = [-1]
B2) mySingle = [12.34], yourSingle = [12.34]
B3) mySingle = [], yourSingle = [-333.33]

*/
```

GetValueOrDefault()

Retrieves the value of the current Nullable<T> object, or the default value of the underlying type.

```
C#

public readonly T GetValueOrDefault ();
```

Returns

T

The value of the Value property if the HasValue property is `true`; otherwise, the default value of the underlying type.

Remarks

The GetValueOrDefault method returns a value even if the HasValue property is `false` (unlike the Value property, which throws an exception). If the HasValue property is `false`, the method returns the default value of the underlying type.

See also

- GetValueOrDefault(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetValueOrDefault(T)

Retrieves the value of the current Nullable<T> object, or the specified default value.

C#

```
public readonly T GetValueOrDefault (T defaultValue);
```

Parameters

`defaultValue` T

A value to return if the HasValue property is `false`.

Returns

T

The value of the Value property if the HasValue property is `true`; otherwise, the `defaultValue` parameter.

Remarks

The GetValueOrDefault method returns a value even if the HasValue property is `false` (unlike the Value property, which throws an exception).

See also

- GetValueOrDefault()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the text representation of the value of the current Nullable<T> object.

C#

```
public override string? ToString ();
```

Returns

String

The text representation of the value of the current Nullable<T> object if the HasValue property is `true`, or an empty string ("") if the HasValue property is `false`.

Examples

The following code example displays the value of the current Nullable<T> object.

C#

```
// This code example demonstrates the
// Nullable<T>.ToString method.

using System;

class Sample
{
    public static void Main()
    {
    DateTime? nullableDate;

// Display the current date and time.
    nullableDate = DateTime.Now;
    Display("1)", nullableDate);

// Assign null (Nothing in Visual Basic) to nullableDate, then
// display its value.
```

```
    nullableDate = null;
    Display("2)", nullableDate);
    }

// Display the text representation of a nullable DateTime.
    public static void Display(string title, DateTime? dspDT)
    {
    string msg = dspDT.ToString();

    Console.Write("{0} ", title);
    if (String.IsNullOrEmpty(msg))
        Console.WriteLine("The nullable DateTime has no defined value.");
    else
        Console.WriteLine("The current date and time is {0}.", msg);
    }
}

/*
This code example produces the following results:

1) The current date and time is 4/19/2005 8:28:14 PM.
2) The nullable DateTime has no defined value.

*/
```

Remarks

The ToString property returns the string yielded by calling the `ToString` property of the object returned by the Value property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Nullable<T>.Explicit(Nullable<T> to T) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an explicit conversion of a Nullable<T> instance to its underlying value.

```
C#

public static explicit operator T (T? value);
```

Parameters

value Nullable<T>

A nullable value.

Returns

T

The value of the Value property for the `value` parameter.

Examples

The Explicit operator enables code such as the following, which converts a `Nullable(Of Int32)` value to an Int32 value.

```
C#

using System;

public class Example
{
   public static void Main()
   {
       var nullInt = new Nullable<int>(172);
       // Convert with CInt conversion method.
```

```
        Console.WriteLine((int)nullInt);
        // Convert with Convert.ChangeType.
        Console.WriteLine(Convert.ChangeType(nullInt, typeof(int)));
    }
}
// The example displays the following output:
//       172
//       172
```

Remarks

This operator supports the explicit conversion of the current Nullable<T> instance to type `T`, the type of Value. The syntax for such explicit conversions is language-dependent. You may also be able to perform the conversion by calling the Convert.ChangeType method.

The equivalent method for this operator is Nullable<T>.Value

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Nullable<T>.Implicit(T to Nullable<T>) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new Nullable<T> object initialized to a specified value.

C#

```
public static implicit operator T? (T value);
```

Parameters

`value` T

A value type.

Returns

Nullable<T>

A Nullable<T> object whose Value property is initialized with the `value` parameter.

Remarks

The Value property of the new Nullable<T> value is initialized to the `value` parameter and the HasValue property is initialized to `true`.

The equivalent method for this operator is Nullable<T>.Nullable<T>(T)

> **Note**
>
> In C# and Visual Basic, an implicit conversion from `T` to **Nullable<T>** does not invoke this operator because these languages have special rules for the conversion.

This operator is provided for the benefit of languages that do not have such special rules.

> **Note**
>
> In C# and Visual Basic, an implicit conversion from a `null` or `Nothing` literal to **Nullable<T>** produces a **Nullable<T>** value whose **HasValue** property is initialized to `false`. The conversion occurs at compile time in these languages and does not invoke any operator.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NullReferenceException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to dereference a null object reference.

C#

```
public class NullReferenceException : SystemException
```

Inheritance Object → Exception → SystemException → NullReferenceException

Remarks

A NullReferenceException exception is thrown when you try to access a member on a type whose value is `null`. A NullReferenceException exception typically reflects developer error and is thrown in the following scenarios:

- You've forgotten to instantiate a reference type. In the following example, `names` is declared but never instantiated:

  C#

  ```
  using System;
  using System.Collections.Generic;

  public class Example
  {
     public static void Main(string[] args)
     {
        int value = Int32.Parse(args[0]);
        List<String> names;
        if (value > 0)
           names = new List<String>();

        names.Add("Major Major Major");
     }
  }
  // Compilation displays a warning like the following:
  ```

```
//    Example1.cs(10) : warning BC42104: Variable //names// is used
before it
//    has been assigned a value. A null reference exception could
result
//    at runtime.
//
//          names.Add("Major Major Major")
//          ~~~~~
// The example displays output like the following output:
//    Unhandled Exception: System.NullReferenceException: Object
reference
//    not set to an instance of an object.
//       at Example.Main()
```

Some compilers issue a warning when they compile this code. Others issue an error, and the compilation fails. To address this problem, instantiate the object so that its value is no longer `null`. The following example does this by calling a type's class constructor.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      List<String> names = new List<String>();
      names.Add("Major Major Major");
   }
}
```

- You've forgotten to dimension an array before initializing it. In the following example, `values` is declared to be an integer array, but the number of elements that it contains is never specified. The attempt to initialize its values therefore thrown a NullReferenceException exception.

C#

```
using System;

public class Example
{
   public static void Main()
   {
       int[] values = null;
       for (int ctr = 0; ctr <= 9; ctr++)
          values[ctr] = ctr * 2;
```

```
        foreach (var value in values)
           Console.WriteLine(value);
    }
}
// The example displays the following output:
//    Unhandled Exception:
//       System.NullReferenceException: Object reference not set to an
instance of an object.
//       at Example.Main()
```

You can eliminate the exception by declaring the number of elements in the array before initializing it, as the following example does.

C#

```
using System;

public class Example
{
   public static void Main()
   {
       int[] values = new int[10];
       for (int ctr = 0; ctr <= 9; ctr++)
          values[ctr] = ctr * 2;

       foreach (var value in values)
          Console.WriteLine(value);
   }
}
// The example displays the following output:
//    0
//    2
//    4
//    6
//    8
//    10
//    12
//    14
//    16
//    18
```

For more information on declaring and initializing arrays, see Arrays and Arrays.

- You get a **null** return value from a method, and then call a method on the returned type. This sometimes is the result of a documentation error; the documentation fails to note that a method call can return `null`. In other cases, your code erroneously assumes that the method will always return a non-**null** value.

The code in the following example assumes that the Array.Find method always returns `Person` object whose `FirstName` field matches a search string. Because there is no match, the runtime throws a NullReferenceException exception.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Person[] persons = Person.AddRange( new String[] { "Abigail", "Abra",
                                                "Abraham", "Adrian", "Ariella",
                                                "Arnold", "Aston", "Astor" } );
      String nameToFind = "Robert";
      Person found = Array.Find(persons, p => p.FirstName == nameToFind);
      Console.WriteLine(found.FirstName);
   }
}

public class Person
{
   public static Person[] AddRange(String[] firstNames)
   {
      Person[] p = new Person[firstNames.Length];
      for (int ctr = 0; ctr < firstNames.Length; ctr++)
         p[ctr] = new Person(firstNames[ctr]);

      return p;
   }

   public Person(String firstName)
   {
      this.FirstName = firstName;
   }

   public String FirstName;
}
// The example displays the following output:
//       Unhandled Exception: System.NullReferenceException:
//       Object reference not set to an instance of an object.
//          at Example.Main()
```

To address this problem, test the method's return value to ensure that it is not `null` before calling any of its members, as the following example does.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Person[] persons = Person.AddRange( new String[] { "Abigail",
"Abra",
                                          "Abraham", "Adrian",
"Ariella",
                                          "Arnold", "Aston", "Astor" }
);
      String nameToFind = "Robert";
      Person found = Array.Find(persons, p => p.FirstName ==
nameToFind);
      if (found != null)
         Console.WriteLine(found.FirstName);
      else
         Console.WriteLine("{0} not found.", nameToFind);
   }
}

public class Person
{
   public static Person[] AddRange(String[] firstNames)
   {
      Person[] p = new Person[firstNames.Length];
      for (int ctr = 0; ctr < firstNames.Length; ctr++)
         p[ctr] = new Person(firstNames[ctr]);

      return p;
   }

   public Person(String firstName)
   {
      this.FirstName = firstName;
   }

   public String FirstName;
}
// The example displays the following output:
//        Robert not found
```

- You're using an expression (for example, you're chaining a list of methods or properties together) to retrieve a value and, although you're checking whether the value is `null`, the runtime still throws a NullReferenceException exception. This occurs because one of the intermediate values in the expression returns `null`. As a result, your test for `null` is never evaluated.

The following example defines a `Pages` object that caches information about web pages, which are presented by `Page` objects. The `Example.Main` method checks whether the current web page has a non-null title and, if it does, displays the title. Despite this check, however, the method throws a NullReferenceException exception.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      var pages = new Pages();
      if (! String.IsNullOrEmpty(pages.CurrentPage.Title)) {
         String title = pages.CurrentPage.Title;
         Console.WriteLine("Current title: '{0}'", title);
      }
   }
}

public class Pages
{
   Page[] page = new Page[10];
   int ctr = 0;

   public Page CurrentPage
   {
      get { return page[ctr]; }
      set {
         // Move all the page objects down to accommodate the new one.
         if (ctr > page.GetUpperBound(0)) {
            for (int ndx = 1; ndx <= page.GetUpperBound(0); ndx++)
               page[ndx - 1] = page[ndx];
         }
         page[ctr] = value;
         if (ctr < page.GetUpperBound(0))
            ctr++;
      }
   }

   public Page PreviousPage
   {
      get {
         if (ctr == 0) {
            if (page[0] == null)
               return null;
            else
               return page[0];
         }
         else {
```

```
            ctr--;
            return page[ctr + 1];
         }
      }
   }
}

public class Page
{
   public Uri URL;
   public String Title;
}
// The example displays the following output:
//    Unhandled Exception:
//       System.NullReferenceException: Object reference not set to an
instance of an object.
//       at Example.Main()
```

The exception is thrown because `pages.CurrentPage` returns `null` if no page information is stored in the cache. This exception can be corrected by testing the value of the `CurrentPage` property before retrieving the current `Page` object's `Title` property, as the following example does:

C#

```
using System;

public class Example
{
   public static void Main()
   {
      var pages = new Pages();
      Page current = pages.CurrentPage;
      if (current != null) {
         String title = current.Title;
         Console.WriteLine("Current title: '{0}'", title);
      }
      else {
         Console.WriteLine("There is no page information in the
cache.");
      }
   }
}
// The example displays the following output:
//       There is no page information in the cache.
```

- You're enumerating the elements of an array that contains reference types, and your attempt to process one of the elements throws a NullReferenceException exception.

The following example defines a string array. A `for` statement enumerates the elements in the array and calls each string's Trim method before displaying the string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] values = { "one", null, "two" };
      for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
         Console.Write("{0}{1}", values[ctr].Trim(),
                        ctr == values.GetUpperBound(0) ? "" : ", ");
      Console.WriteLine();
   }
}
// The example displays the following output:
//    Unhandled Exception:
//       System.NullReferenceException: Object reference not set to an
instance of an object.
//       at Example.Main()
```

This exception occurs if you assume that each element of the array must contain a non-null value, and the value of the array element is in fact `null`. The exception can be eliminated by testing whether the element is `null` before performing any operation on that element, as the following example shows.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] values = { "one", null, "two" };
      for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
         Console.Write("{0}{1}",
                        values[ctr] != null ? values[ctr].Trim() : "",
                        ctr == values.GetUpperBound(0) ? "" : ", ");
      Console.WriteLine();
   }
}
// The example displays the following output:
//       one, , two
```

- A NullReferenceException exception can be thrown by a method when it accesses a member of one of its arguments, but that argument is `null`. The `PopulateNames` method in the following example throws the exception at the line `names.Add(arrName);`.

  C#

  ```
  using System;
  using System.Collections.Generic;

  public class Example
  {
     public static void Main()
     {
        List<String> names = GetData();
        PopulateNames(names);
     }

     private static void PopulateNames(List<String> names)
     {
        String[] arrNames = { "Dakota", "Samuel", "Nikita",
                              "Koani", "Saya", "Yiska", "Yumaevsky" };
        foreach (var arrName in arrNames)
           names.Add(arrName);
     }

     private static List<String> GetData()
     {
        return null;
     }
  }
  // The example displays output like the following:
  //    Unhandled Exception: System.NullReferenceException: Object
  reference
  //    not set to an instance of an object.
  //       at Example.PopulateNames(List`1 names)
  //       at Example.Main()
  ```

  To address this issue, make sure that the argument passed to the method is not `null`, or handle the thrown exception in a `try…catch…finally` block. For more information, see Exceptions.

The following Microsoft intermediate language (MSIL) instructions throw NullReferenceException: `callvirt`, `cpblk`, `cpobj`, `initblk`, `ldelem.<type>`, `ldelema`, `ldfld`, `ldflda`, `ldind.<type>`, `ldlen`, `stelem.<type>`, `stfld`, `stind.<type>`, `throw`, and `unbox`.

NullReferenceException uses the HRESULT COR_E_NULLREFERENCE, which has the value 0x80004003.

For a list of initial property values for an instance of NullReferenceException, see the NullReferenceException constructors.

Handling NullReferenceException in release code

It's usually better to avoid a NullReferenceException than to handle it after it occurs. Handling an exception can make your code harder to maintain and understand, and can sometimes introduce other bugs. A NullReferenceException is often a non-recoverable error. In these cases, letting the exception stop the app might be the best alternative.

However, there are many situations where handling the error can be useful:

- Your app can ignore objects that are null. For example, if your app retrieves and processes records in a database, you might be able to ignore some number of bad records that result in null objects. Recording the bad data in a log file or in the application UI might be all you have to do.
- You can recover from the exception. For example, a call to a web service that returns a reference type might return null if the connection is lost or the connection times out. You can attempt to reestablish the connection and try the call again.
- You can restore the state of your app to a valid state. For example, you might be performing a multi-step task that requires you to save information to a data store before you call a method that throws a NullReferenceException. If the uninitialized object would corrupt the data record, you can remove the previous data before you close the app.
- You want to report the exception. For example, if the error was caused by a mistake from the user of your app, you can generate a message to help them supply the correct information. You can also log information about the error to help you fix the problem. Some frameworks, like ASP.NET, have a high-level exception handler that captures all errors to that the app never crashes; in that case, logging the exception might be the only way you can know that it occurs.

Constructors

Expand table

NullReferenceException()	Initializes a new instance of the NullReferenceException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The value

	'null' was found where an instance of an object was required." This message takes into account the current system culture.
NullReference Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NullReferenceException class with serialized data.
NullReference Exception(String)	Initializes a new instance of the NullReferenceException class with a specified error message.
NullReference Exception(String, Exception)	Initializes a new instance of the NullReferenceException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

NullReferenceException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the NullReferenceException class.

Overloads

Expand table

NullReferenceException()	Initializes a new instance of the NullReferenceException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The value 'null' was found where an instance of an object was required." This message takes into account the current system culture.
NullReferenceException(String)	Initializes a new instance of the NullReferenceException class with a specified error message.
NullReferenceException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the NullReferenceException class with serialized data.
NullReferenceException(String, Exception)	Initializes a new instance of the NullReferenceException class with a specified error message and a reference to the inner exception that is the cause of this exception.

NullReferenceException()

Initializes a new instance of the NullReferenceException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The value 'null' was found where an instance of an object was required." This message takes into account the current system culture.

```
C#
public NullReferenceException ();
```

Remarks

The following table shows the initial property values for an instance of NullReferenceException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NullReferenceException(String)

Initializes a new instance of the NullReferenceException class with a specified error message.

```
C#
public NullReferenceException (string? message);
```

Parameters

`message` String

A String that describes the error. The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of NullReferenceException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

NullReferenceException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the NullReferenceException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected NullReferenceException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

NullReferenceException(String, Exception)

Initializes a new instance of the NullReferenceException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public NullReferenceException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of NullReferenceException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Object Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Supports all classes in the .NET class hierarchy and provides low-level services to derived classes. This is the ultimate base class of all .NET classes; it is the root of the type hierarchy.

C#

```
public class Object
```

Examples

The following example defines a Point type derived from the Object class and overrides many of the virtual methods of the Object class. In addition, the example shows how to call many of the static and instance methods of the Object class.

C#

```
using System;

// The Point class is derived from System.Object.
class Point
{
    public int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public override bool Equals(object obj)
    {
        // If this and obj do not refer to the same type, then they are not
equal.
        if (obj.GetType() != this.GetType()) return false;

        // Return true if  x and y fields match.
        var other = (Point) obj;
```

```
        return (this.x == other.x) && (this.y == other.y);
    }

    // Return the XOR of the x and y fields.
    public override int GetHashCode()
    {
        return x ^ y;
    }

    // Return the point's value as a string.
    public override String ToString()
    {
        return $"({x}, {y})";
    }

    // Return a copy of this point object by making a simple field copy.
    public Point Copy()
    {
        return (Point) this.MemberwiseClone();
    }
}

public sealed class App
{
    static void Main()
    {
        // Construct a Point object.
        var p1 = new Point(1,2);

        // Make another Point object that is a copy of the first.
        var p2 = p1.Copy();

        // Make another variable that references the first Point object.
        var p3 = p1;

        // The line below displays false because p1 and p2 refer to two
different objects.
        Console.WriteLine(Object.ReferenceEquals(p1, p2));

        // The line below displays true because p1 and p2 refer to two
different objects that have the same value.
        Console.WriteLine(Object.Equals(p1, p2));

        // The line below displays true because p1 and p3 refer to one
object.
        Console.WriteLine(Object.ReferenceEquals(p1, p3));

        // The line below displays: p1's value is: (1, 2)
        Console.WriteLine($"p1's value is: {p1.ToString()}");
    }
}

// This code example produces the following output:
//
// False
```

```
// True
// True
// p1's value is: (1, 2)
//
```

Remarks

For more information about this API, see Supplemental API remarks for Object.

Constructors

Expand table

Object()	Initializes a new instance of the Object class.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.
Equals(Object, Object)	Determines whether the specified object instances are considered equal.
Finalize()	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
GetHashCode()	Serves as the default hash function.
GetType()	Gets the Type of the current instance.
MemberwiseClone()	Creates a shallow copy of the current Object.
ReferenceEquals(Object, Object)	Determines whether the specified Object instances are the same instance.
ToString()	Returns a string that represents the current object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

Public static (`Shared` in Visual Basic) members of this type are thread safe. Instance members are not guaranteed to be thread-safe.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Object Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Object class.

C#

```
public Object ();
```

Remarks

This constructor is called by constructors in derived classes, but it can also be used to directly create an instance of the Object class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Object.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two object instances are equal.

Overloads

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object.
Equals(Object, Object)	Determines whether the specified object instances are considered equal.

Equals(Object)

Determines whether the specified object is equal to the current object.

C#

```
public virtual bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with the current object.

Returns

Boolean

`true` if the specified object is equal to the current object; otherwise, `false`.

Remarks

For more information about this API, see Supplemental API remarks for Object.Equals.

See also

- ReferenceEquals(Object, Object)
- GetHashCode()
- ValueType
- IComparable
- ArrayList
- Hashtable

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object, Object)

Determines whether the specified object instances are considered equal.

C#

```
public static bool Equals (object? objA, object? objB);
```

Parameters

objA Object

The first object to compare.

objB Object

The second object to compare.

Returns

Boolean

`true` if the objects are considered equal; otherwise, `false`. If both `objA` and `objB` are **null**, the method returns `true`.

Examples

The following example illustrates the Equals(Object, Object) method and compares it with the ReferenceEquals method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Dog m1 = new Dog("Alaskan Malamute");
      Dog m2 = new Dog("Alaskan Malamute");
      Dog g1 = new Dog("Great Pyrenees");
      Dog g2 = g1;
      Dog d1 = new Dog("Dalmation");
      Dog n1 = null;
      Dog n2 = null;

      Console.WriteLine("null = null: {0}", Object.Equals(n1, n2));
      Console.WriteLine("null Reference Equals null: {0}\n",
Object.ReferenceEquals(n1, n2));

      Console.WriteLine("{0} = {1}: {2}", g1, g2, Object.Equals(g1, g2));
      Console.WriteLine("{0} Reference Equals {1}: {2}\n", g1, g2,
Object.ReferenceEquals(g1, g2));

      Console.WriteLine("{0} = {1}: {2}", m1, m2, Object.Equals(m1, m2));
      Console.WriteLine("{0} Reference Equals {1}: {2}\n", m1, m2,
Object.ReferenceEquals(m1, m2));

      Console.WriteLine("{0} = {1}: {2}", m1, d1, Object.Equals(m1, d1));
      Console.WriteLine("{0} Reference Equals {1}: {2}", m1, d1,
Object.ReferenceEquals(m1, d1));
   }
}

public class Dog
{
```

```
    // Public field.
    public string Breed;

    // Class constructor.
    public Dog(string dogBreed)
    {
       this.Breed = dogBreed;
    }

    public override bool Equals(Object obj)
    {
       if (obj == null || !(obj is Dog))
          return false;
       else
          return this.Breed == ((Dog) obj).Breed;
    }

    public override int GetHashCode()
    {
       return this.Breed.GetHashCode();
    }

    public override string ToString()
    {
       return this.Breed;
    }
 }
 // The example displays the following output:
 //       null = null: True
 //       null Reference Equals null: True
 //
 //       Great Pyrenees = Great Pyrenees: True
 //       Great Pyrenees Reference Equals Great Pyrenees: True
 //
 //       Alaskan Malamute = Alaskan Malamute: True
 //       Alaskan Malamute Reference Equals Alaskan Malamute: False
 //
 //       Alaskan Malamute = Dalmation: False
 //       Alaskan Malamute Reference Equals Dalmation: False
```

Remarks

The static Equals(Object, Object) method indicates whether two objects, `objA` and `objB`, are equal. It also enables you to test objects whose value is **null** for equality. It compares `objA` and `objB` for equality as follows:

- It determines whether the two objects represent the same object reference. If they do, the method returns `true`. This test is equivalent to calling the ReferenceEquals method. In addition, if both `objA` and `objB` are **null**, the method returns `true`.

- It determines whether either `objA` or `objB` is **null**. If so, it returns `false`.
- If the two objects do not represent the same object reference and neither is **null**, it calls `objA.Equals(objB)` and returns the result. This means that if `objA` overrides the Object.Equals(Object) method, this override is called.

See also

- Equals(Object)
- ReferenceEquals(Object, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Object.Finalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.

C#

```
~Object ();
```

Examples

The following example verifies that the Finalize method is called when an object that overrides Finalize is destroyed. Note that, in a production application, the Finalize method would be overridden to release unmanaged resources held by the object. Also note that the C# example provides a destructor instead of overriding the Finalize method.

C#

```
using System;
using System.Diagnostics;

public class ExampleClass
{
   Stopwatch sw;

   public ExampleClass()
   {
      sw = Stopwatch.StartNew();
      Console.WriteLine("Instantiated object");
   }

   public void ShowDuration()
   {
      Console.WriteLine("This instance of {0} has been in existence for
{1}",
                        this, sw.Elapsed);
   }
```

```
   ~ExampleClass()
   {
      Console.WriteLine("Finalizing object");
      sw.Stop();
      Console.WriteLine("This instance of {0} has been in existence for
{1}",
                        this, sw.Elapsed);
   }
}

public class Demo
{
   public static void Main()
   {
      ExampleClass ex = new ExampleClass();
      ex.ShowDuration();
   }
}
// The example displays output like the following:
//    Instantiated object
//    This instance of ExampleClass has been in existence for
00:00:00.0011060
//    Finalizing object
//    This instance of ExampleClass has been in existence for
00:00:00.0036294
```

For an additional example that overrides the Finalize method, see the GC.SuppressFinalize method.

Remarks

For more information about this API, see Supplemental API remarks for Object.Finalize.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- SuppressFinalize(Object)
- ReRegisterForFinalize(Object)
- WaitForPendingFinalizers()
- WeakReference

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Object.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as the default hash function.

```
C#
public virtual int GetHashCode ();
```

Returns

Int32

A hash code for the current object.

Remarks

For more information about this API, see Supplemental API remarks for Object.GetHashCode.

Notes to Inheritors

A hash function is used to quickly generate a number (hash code) that corresponds to the value of an object. Hash functions are usually specific to each type and, for uniqueness, must use at least one of the instance fields as input. Hash codes should not be computed by using the values of static fields.

For classes derived from Object, the `GetHashCode` method can delegate to the base class GetHashCode() implementation only if the derived class defines equality to be reference equality. The default implementation of GetHashCode() for reference types returns a hash code that is equivalent to the one returned by the GetHashCode(Object) method. You can override GetHashCode() for immutable reference types. In general, for mutable reference types, you should override GetHashCode() only if:

- You can compute the hash code from fields that are not mutable; or

- You can ensure that the hash code of a mutable object does not change while the object is contained in a collection that relies on its hash code.

Otherwise, you might think that the mutable object is lost in the hash table. If you do choose to override GetHashCode() for a mutable reference type, your documentation should make it clear that users of your type should not modify object values while the object is stored in a hash table.

For value types, GetHashCode() provides a default hash code implementation that uses reflection. You should consider overriding it for better performance.

For more information and examples that compute hash codes in a variety of ways, see the Examples section.

A hash function must have the following properties:

- If two objects compare as equal, the GetHashCode() method for each object must return the same value. However, if two objects do not compare as equal, the GetHashCode() methods for the two objects do not have to return different values.

- The GetHashCode() method for an object must consistently return the same hash code as long as there is no modification to the object state that determines the return value of the object's System.Object.Equals method. Note that this is true only for the current execution of an application, and that a different hash code can be returned if the application is run again.

- For the best performance, a hash function should generate an even distribution for all input, including input that is heavily clustered. An implication is that small modifications to object state should result in large modifications to the resulting hash code for best hash table performance.

- Hash functions should be inexpensive to compute.

- The GetHashCode() method should not throw exceptions.

For example, the implementation of the GetHashCode() method provided by the String class returns identical hash codes for identical string values. Therefore, two String objects return the same hash code if they represent the same string value. Also, the method uses all the characters in the string to generate reasonably randomly distributed output, even when the input is clustered in certain ranges (for example, many users might have strings that contain only the lower 128 ASCII characters, even though a string can contain any of the 65,535 Unicode characters).

Providing a good hash function on a class can significantly affect the performance of adding those objects to a hash table. In a hash table with keys that provide a good

implementation of a hash function, searching for an element takes constant time (for example, an O(1) operation). In a hash table with a poor implementation of a hash function, the performance of a search depends on the number of items in the hash table (for example, an O(`n`) operation, where `n` is the number of items in the hash table). A malicious user can input data that increases the number of collisions, which can significantly degrade the performance of applications that depend on hash tables, under the following conditions:

- When hash functions produce frequent collisions.
- When a large proportion of objects in a hash table produce hash codes that are equal or approximately equal to one another.
- When users input the data from which the hash code is computed.

Derived classes that override GetHashCode() must also override Equals(Object) to guarantee that two objects considered equal have the same hash code; otherwise, the Hashtable type might not work correctly.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Hashtable
- Equals(Object)
- GetHashCode(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Object.GetType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Type of the current instance.

```
C#

public Type GetType ();
```

Returns

Type

The exact runtime type of the current instance.

Examples

The following code example demonstrates that GetType returns the runtime type of the current instance.

```
C#

using System;

public class MyBaseClass {
}

public class MyDerivedClass: MyBaseClass {
}

public class Test
{
   public static void Main()
   {
      MyBaseClass myBase = new MyBaseClass();
      MyDerivedClass myDerived = new MyDerivedClass();
      object o = myDerived;
      MyBaseClass b = myDerived;
```

```
        Console.WriteLine("mybase: Type is {0}", myBase.GetType());
        Console.WriteLine("myDerived: Type is {0}", myDerived.GetType());
        Console.WriteLine("object o = myDerived: Type is {0}", o.GetType());
        Console.WriteLine("MyBaseClass b = myDerived: Type is {0}",
b.GetType());
    }
}
// The example displays the following output:
//    mybase: Type is MyBaseClass
//    myDerived: Type is MyDerivedClass
//    object o = myDerived: Type is MyDerivedClass
//    MyBaseClass b = myDerived: Type is MyDerivedClass
```

Remarks

Because System.Object is the base class for all types in the .NET type system, the GetType method can be used to return Type objects that represent all .NET types. .NET recognizes the following five categories of types:

- Classes, which are derived from System.Object,
- Value types, which are derived from System.ValueType.
- Interfaces, which are derived from System.Object starting with the .NET Framework 2.0.
- Enumerations, which are derived from System.Enum.
- Delegates, which are derived from System.MulticastDelegate.

For two objects `x` and `y` that have identical runtime types, `Object.ReferenceEquals(x.GetType(),y.GetType())` returns `true`. The following example uses the GetType method with the ReferenceEquals method to determine whether one numeric value is the same type as two other numeric values.

C#

```
int n1 = 12;
int n2 = 82;
long n3 = 12;

Console.WriteLine("n1 and n2 are the same type: {0}",
                  Object.ReferenceEquals(n1.GetType(), n2.GetType()));
Console.WriteLine("n1 and n3 are the same type: {0}",
                  Object.ReferenceEquals(n1.GetType(), n3.GetType()));

// The example displays the following output:
```

```
//       n1 and n2 are the same type: True
//       n1 and n3 are the same type: False
```

> **Note**
>
> To determine whether an object is a specific type, you can use your language's type comparison keyword or construct. For example, you can use the `TypeOf…Is` construct in Visual Basic or the `is` keyword in C#.

The GetType method is inherited by all types that derive from Object. This means that, in addition to using your own language's comparison keyword, you can use the GetType method to determine the type of a particular object, as the following example shows.

C#

```
object[] values = { (int) 12, (long) 10653, (byte) 12, (sbyte) -5,
                    16.3, "string" };
foreach (var value in values) {
   Type t = value.GetType();
   if (t.Equals(typeof(byte)))
      Console.WriteLine("{0} is an unsigned byte.", value);
   else if (t.Equals(typeof(sbyte)))
      Console.WriteLine("{0} is a signed byte.", value);
   else if (t.Equals(typeof(int)))
      Console.WriteLine("{0} is a 32-bit integer.", value);
   else if (t.Equals(typeof(long)))
      Console.WriteLine("{0} is a 64-bit integer.", value);
   else if (t.Equals(typeof(double)))
      Console.WriteLine("{0} is a double-precision floating point.",
                        value);
   else
      Console.WriteLine("'{0}' is another data type.", value);
}

// The example displays the following output:
//    12 is a 32-bit integer.
//    10653 is a 32-bit integer.
//    12 is an unsigned byte.
//    -5 is a signed byte.
//    16.3 is a double-precision floating point.
//    'string' is another data type.
```

The Type object exposes the metadata associated with the class of the current Object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Type

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Object.MemberwiseClone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a shallow copy of the current Object.

C#

```
protected object MemberwiseClone ();
```

Returns

Object

A shallow copy of the current Object.

Examples

The following example illustrates the MemberwiseClone method. It defines a `ShallowCopy` method that calls the MemberwiseClone method to perform a shallow copy operation on a `Person` object. It also defines a `DeepCopy` method that performs a deep copy operation on a `Person` object.

C#

```
using System;

public class IdInfo
{
    public int IdNumber;

    public IdInfo(int IdNumber)
    {
        this.IdNumber = IdNumber;
    }
}

public class Person
{
```

```
    public int Age;
    public string Name;
    public IdInfo IdInfo;

    public Person ShallowCopy()
    {
       return (Person) this.MemberwiseClone();
    }

    public Person DeepCopy()
    {
       Person other = (Person) this.MemberwiseClone();
       other.IdInfo = new IdInfo(IdInfo.IdNumber);
       other.Name = String.Copy(Name);
       return other;
    }
}

public class Example
{
    public static void Main()
    {
        // Create an instance of Person and assign values to its fields.
        Person p1 = new Person();
        p1.Age = 42;
        p1.Name = "Sam";
        p1.IdInfo = new IdInfo(6565);

        // Perform a shallow copy of p1 and assign it to p2.
        Person p2 = p1.ShallowCopy();

        // Display values of p1, p2
        Console.WriteLine("Original values of p1 and p2:");
        Console.WriteLine("   p1 instance values: ");
        DisplayValues(p1);
        Console.WriteLine("   p2 instance values:");
        DisplayValues(p2);

        // Change the value of p1 properties and display the values of p1
and p2.
        p1.Age = 32;
        p1.Name = "Frank";
        p1.IdInfo.IdNumber = 7878;
        Console.WriteLine("\nValues of p1 and p2 after changes to p1:");
        Console.WriteLine("   p1 instance values: ");
        DisplayValues(p1);
        Console.WriteLine("   p2 instance values:");
        DisplayValues(p2);

        // Make a deep copy of p1 and assign it to p3.
        Person p3 = p1.DeepCopy();
        // Change the members of the p1 class to new values to show the deep
copy.
        p1.Name = "George";
        p1.Age = 39;
```

```
        p1.IdInfo.IdNumber = 8641;
        Console.WriteLine("\nValues of p1 and p3 after changes to p1:");
        Console.WriteLine("   p1 instance values: ");
        DisplayValues(p1);
        Console.WriteLine("   p3 instance values:");
        DisplayValues(p3);
    }

    public static void DisplayValues(Person p)
    {
        Console.WriteLine("      Name: {0:s}, Age: {1:d}", p.Name, p.Age);
        Console.WriteLine("      Value: {0:d}", p.IdInfo.IdNumber);
    }
}
// The example displays the following output:
//       Original values of p1 and p2:
//          p1 instance values:
//             Name: Sam, Age: 42
//             Value: 6565
//          p2 instance values:
//             Name: Sam, Age: 42
//             Value: 6565
//
//       Values of p1 and p2 after changes to p1:
//          p1 instance values:
//             Name: Frank, Age: 32
//             Value: 7878
//          p2 instance values:
//             Name: Sam, Age: 42
//             Value: 7878
//
//       Values of p1 and p3 after changes to p1:
//          p1 instance values:
//             Name: George, Age: 39
//             Value: 8641
//          p3 instance values:
//             Name: Frank, Age: 32
//             Value: 7878
```

In this example, the `Person.IdInfo` property returns an `IdInfo` object. As the output from the example shows, when a `Person` object is cloned by calling the MemberwiseClone method, the cloned `Person` object is an independent copy of the original object, except that they share the same `Person.IdInfo` object reference. As a result, modifying the clone's `Person.IdInfo` property changes the original object's `Person.IdInfo` property. On the other hand, when a deep copy operation is performed, the cloned `Person` object, including its `Person.IdInfo` property, can be modified without affecting the original object.

Remarks

The MemberwiseClone method creates a shallow copy by creating a new object, and then copying the nonstatic fields of the current object to the new object. If a field is a value type, a bit-by-bit copy of the field is performed. If a field is a reference type, the reference is copied but the referred object is not; therefore, the original object and its clone refer to the same object.

For example, consider an object called X that references objects A and B. Object B, in turn, references object C. A shallow copy of X creates new object X2 that also references objects A and B. In contrast, a deep copy of X creates a new object X2 that references the new objects A2 and B2, which are copies of A and B. B2, in turn, references the new object C2, which is a copy of C. The example illustrates the difference between a shallow and a deep copy operation.

There are numerous ways to implement a deep copy operation if the shallow copy operation performed by the MemberwiseClone method does not meet your needs. These include the following:

- Call a class constructor of the object to be copied to create a second object with property values taken from the first object. This assumes that the values of an object are entirely defined by its class constructor.
- Call the MemberwiseClone method to create a shallow copy of an object, and then assign new objects whose values are the same as the original object to any properties or fields whose values are reference types. The `DeepCopy` method in the example illustrates this approach.
- Serialize the object to be deep copied, and then restore the serialized data to a different object variable.
- Use reflection with recursion to perform the deep copy operation.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Object.ReferenceEquals(Object, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified Object instances are the same instance.

C#

```
public static bool ReferenceEquals (object? objA, object? objB);
```

Parameters

objA Object

The first object to compare.

objB Object

The second object to compare.

Returns

Boolean

`true` if `objA` is the same instance as `objB` or if both are **null**; otherwise, `false`.

Examples

The following example uses ReferenceEquals to determine if two objects are the same instance.

C#

```
object o = null;
object p = null;
object q = new Object();
```

```
Console.WriteLine(Object.ReferenceEquals(o, p));
p = q;
Console.WriteLine(Object.ReferenceEquals(p, q));
Console.WriteLine(Object.ReferenceEquals(o, p));

// This code produces the following output:
//   True
//   True
//   False
```

Remarks

Unlike the Equals method and the equality operator, the ReferenceEquals method cannot be overridden. Because of this, if you want to test two object references for equality and you are unsure about the implementation of the `Equals` method, you can call the ReferenceEquals method.

However, the return value of the ReferenceEquals method may appear to be anomalous in these two scenarios:

- When comparing value types. If `objA` and `objB` are value types, they are boxed before they are passed to the ReferenceEquals method. This means that if both `objA` and `objB` represent the same instance of a value type, the ReferenceEquals method nevertheless returns `false`, as the following example shows.

  C#

  ```
  int int1 = 3;
  Console.WriteLine(Object.ReferenceEquals(int1, int1));
  Console.WriteLine(int1.GetType().IsValueType);

  // The example displays the following output:
  //       False
  //       True
  ```

  For information on boxing value types, see Boxing and Unboxing.

- When comparing strings. If `objA` and `objB` are strings, the ReferenceEquals method returns `true` if the string is interned. It does not perform a test for value equality. In the following example, `s1` and `s2` are equal because they are two instances of a single interned string. However, `s3` and `s4` are not equal, because although they have identical string values, that string is not interned.

  C#

```
String s1 = "String1";
String s2 = "String1";
Console.WriteLine("s1 = s2: {0}", Object.ReferenceEquals(s1, s2));
Console.WriteLine("{0} interned: {1}", s1,
                  String.IsNullOrEmpty(String.IsInterned(s1)) ? "No" :
"Yes");

String suffix = "A";
String s3 = "String" + suffix;
String s4 = "String" + suffix;
Console.WriteLine("s3 = s4: {0}", Object.ReferenceEquals(s3, s4));
Console.WriteLine("{0} interned: {1}", s3,
                  String.IsNullOrEmpty(String.IsInterned(s3)) ? "No" :
"Yes");

// The example displays the following output:
//       s1 = s2: True
//       String1 interned: Yes
//       s3 = s4: False
//       StringA interned: No
```

For more information about string interning, see String.IsInterned.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Object.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the current object.

```
C#
public virtual string? ToString ();
```

Returns

String

A string that represents the current object.

Remarks

For more information about this API, see Supplemental API remarks for Object.ToString.

Notes to Inheritors

When you implement your own types, you should override the ToString() method to return values that are meaningful for those types. Derived classes that require more control over formatting than ToString() provides can implement the IFormattable interface. Its ToString(String, IFormatProvider) method enables you to define format strings that control formatting and to use an IFormatProvider object that can provide for culture-specific formatting.

Overrides of the ToString() method should follow these guidelines:

- The returned string should be friendly and readable by humans.
- The returned string should uniquely identify the value of the object instance.
- The returned string should be as short as possible so that it is suitable for display by a debugger.

- Your ToString() override should not return Empty or a null string.
- Your ToString() override should not throw an exception.
- If the string representation of an instance is culture-sensitive or can be formatted in multiple ways, implement the IFormattable interface.
- If the returned string includes sensitive information, you should first demand an appropriate permission. If the demand succeeds, you can return the sensitive information; otherwise, you should return a string that excludes the sensitive information.
- Your ToString() override should have no observable side effects to avoid complications in debugging. For example, a call to the ToString() method should not change the value of instance fields.
- If your type implements a parsing method (or `Parse` or `TryParse` method, a constructor, or some other static method that instantiates an instance of the type from a string), you should ensure that the string returned by the ToString() method can be converted to an object instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Formatting Types in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ObjectDisposedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an operation is performed on a disposed object.

C#

```
public class ObjectDisposedException : InvalidOperationException
```

Inheritance Object → Exception → SystemException → InvalidOperationException → ObjectDisposedException

Examples

The following example demonstrates an error that causes the `ObjectDisposedException` exception to be thrown.

C#

```
using System;
using System.IO;

public class ObjectDisposedExceptionTest
{
   public static void Main()
   {
      MemoryStream ms = new MemoryStream(16);
      ms.Close();
      try
      {
         ms.ReadByte();
      }
      catch (ObjectDisposedException e)
      {
         Console.WriteLine("Caught: {0}", e.Message);
      }
   }
}
```

This code produces the following output:

```
Caught:
  Cannot access a closed Stream.
```

Remarks

An ObjectDisposedException is thrown when you try to access a member of an object that implements the IDisposable interface or IAsyncDisposable interface, and that object has been disposed. Typically, this exception is caused by one of the following conditions:

- You've called an `IDisposable` object's `Dispose` method (or an `IDisposableAsync` object's `DisposeAsync` method), and you're trying to access an instance member that gets or sets the object's state. The following example illustrates the ObjectDisposedException that is thrown when you try to reset the frequency of timer notifications after you call the Timer.Dispose method.

  C#

  ```
  using System;
  using System.Threading;

  public class Example
  {
     public static void Main()
     {
        Timer t = new Timer(TimerNotification, null,
                            100, Timeout.Infinite);
        Thread.Sleep(2000);
        t.Dispose();

        t.Change(200, 1000);
        Thread.Sleep(3000);
     }

     private static void TimerNotification(Object obj)
     {
        Console.WriteLine("Timer event fired at {0:F}", DateTime.Now);
     }
  }
  // The example displays output like the following:
  //    Timer event fired at Monday, July 14, 2014 11:54:08 AM
  //
  //    Unhandled Exception: System.ObjectDisposedException: Cannot
  access a disposed object.
  //       at System.Threading.TimerQueueTimer.Change(UInt32 dueTime,
  ```

```
UInt32 period)
//       at Example.Main()
```

- You've called an object's `Close` method, and you're trying to access an instance member that gets or sets the object's state. Often, the `Close` method provides a type's public implementation of the IDisposable.Dispose method. The same is true for `CloseAsync` and `<xref:System.IAsyncDisposable.DisposeAsync%2A?displayProperty=nameWithType>`.

- You've called an object's `Dispose` or `DisposeAsync` methods multiple times. Typically, this doesn't throw an exception. However, depending on how a type implements IDisposable.Dispose or IAsyncDisposable.DisposeAsync, it may not allow multiple calls to that method.

In most cases, this exception results from developer error. Instead of handling the error in a `try`/`catch` block, you should correct the error, typically by reinstantiating the object.

Constructors

Expand table

ObjectDisposedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ObjectDisposedException class with serialized data.
ObjectDisposedException(String)	Initializes a new instance of the ObjectDisposedException class with a string containing the name of the disposed object.
ObjectDisposedException(String, Exception)	Initializes a new instance of the ObjectDisposedException class with a specified error message and a reference to the inner exception that is the cause of this exception.
ObjectDisposedException(String, String)	Initializes a new instance of the ObjectDisposedException class with the specified object name and message.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the message that describes the error.
ObjectName	Gets the name of the disposed object.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Retrieves the SerializationInfo object with the parameter name and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)

MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ThrowIf(Boolean, Object)	Throws an ObjectDisposedException if the specified `condition` is `true`.
ThrowIf(Boolean, Type)	Throws an ObjectDisposedException if the specified `condition` is `true`.
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IDisposable

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ObjectDisposedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ObjectDisposedException class.

Overloads

Expand table

ObjectDisposedException(String)	Initializes a new instance of the ObjectDisposedException class with a string containing the name of the disposed object.
ObjectDisposedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the ObjectDisposedException class with serialized data.
ObjectDisposedException(String, Exception)	Initializes a new instance of the ObjectDisposedException class with a specified error message and a reference to the inner exception that is the cause of this exception.
ObjectDisposedException(String, String)	Initializes a new instance of the ObjectDisposedException class with the specified object name and message.

ObjectDisposedException(String)

Initializes a new instance of the ObjectDisposedException class with a string containing the name of the disposed object.

C#

```
public ObjectDisposedException (string? objectName);
```

Parameters

`objectName` String

A string containing the name of the disposed object.

Remarks

The Message property is initialized to a system-supplied message that describes the error and includes the `objectName` parameter. This message takes into account the current system culture.

The `InnerException` property of the new instance is initialized to `null`.

If `objectName` is `null`, the `Message` property contains only an error message.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ObjectDisposedException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the ObjectDisposedException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected ObjectDisposedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ObjectDisposedException(String, Exception)

Initializes a new instance of the ObjectDisposedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public ObjectDisposedException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If `innerException` is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ObjectDisposedException(String, String)

Initializes a new instance of the ObjectDisposedException class with the specified object name and message.

C#

```
public ObjectDisposedException (string? objectName, string? message);
```

Parameters

`objectName` String

The name of the disposed object.

`message` String

The error message that explains the reason for the exception.

Remarks

The `InnerException` property of the new instance is initialized to `null`.

If `objectName` is `null`, the `Message` property contains only an error message.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ObjectDisposedException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the message that describes the error.

```
C#
public override string Message { get; }
```

Property Value

String

A string that describes the error.

Remarks

If the ObjectName property is not `null`, the message includes the name of the object.

This property overrides Exception.Message.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ObjectDisposedException.ObjectName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the disposed object.

```
C#
public string ObjectName { get; }
```

Property Value

String

A string containing the name of the disposed object.

Remarks

If the current property is not `null` or String.Empty, the value of this property is included in the string returned by the Message property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ObjectDisposedException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Retrieves the SerializationInfo object with the parameter name and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ObjectDisposedException.ThrowIf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ThrowIf(Boolean, Object)	Throws an ObjectDisposedException if the specified `condition` is `true`.
ThrowIf(Boolean, Type)	Throws an ObjectDisposedException if the specified `condition` is `true`.

ThrowIf(Boolean, Object)

Throws an ObjectDisposedException if the specified `condition` is `true`.

C#

```
public static void ThrowIf (bool condition, object instance);
```

Parameters

`condition` Boolean

The condition to evaluate.

`instance` Object

The object whose type's full name should be included in any resulting ObjectDisposedException.

Exceptions

ObjectDisposedException

The `condition` is `true`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ThrowIf(Boolean, Type)

Throws an ObjectDisposedException if the specified `condition` is `true`.

C#

```
public static void ThrowIf (bool condition, Type type);
```

Parameters

`condition` Boolean

The condition to evaluate.

`type` Type

The type whose full name should be included in any resulting ObjectDisposedException.

Exceptions

ObjectDisposedException

The `condition` is `true`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ObsoleteAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Marks the program elements that are no longer in use. This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
System.AttributeTargets.Constructor | System.AttributeTargets.Delegate |
System.AttributeTargets.Enum | System.AttributeTargets.Event |
System.AttributeTargets.Field | System.AttributeTargets.Interface |
System.AttributeTargets.Method | System.AttributeTargets.Property |
System.AttributeTargets.Struct, Inherited=false)]
public sealed class ObsoleteAttribute : Attribute
```

Inheritance Object → Attribute → ObsoleteAttribute

Attributes AttributeUsageAttribute

Examples

The following example defines a class that contains a property and a method that are marked with the ObsoleteAttribute attribute. Accessing the value of the `OldProperty` property in code generates a compiler warning, but calling the `CallOldMethod` method generates a compiler error. The example also shows the output that results when you attempt to compile the source code.

C#

```
using System;
using System.Reflection;

public class Example
{
   // Mark OldProperty As Obsolete.
   [ObsoleteAttribute("This property is obsolete. Use NewProperty instead.",
false)]
   public static string OldProperty
```

```
    { get { return "The old property value."; } }

    public static string NewProperty
    { get { return "The new property value."; } }

    // Mark CallOldMethod As Obsolete.
    [ObsoleteAttribute("This method is obsolete. Call CallNewMethod
instead.", true)]
    public static string CallOldMethod()
    {
       return "You have called CallOldMethod.";
    }

    public static string CallNewMethod()
    {
       return "You have called CallNewMethod.";
    }

    public static void Main()
    {
       Console.WriteLine(OldProperty);
       Console.WriteLine();
       Console.WriteLine(CallOldMethod());
    }
}
// The attempt to compile this example produces output like the following
output:
//    Example.cs(31,25): error CS0619: 'Example.CallOldMethod()' is
obsolete:
//            'This method is obsolete. Call CallNewMethod instead.'
//    Example.cs(29,25): warning CS0618: 'Example.OldProperty' is obsolete:
//            'This property is obsolete. Use NewProperty instead.'
```

Remarks

ObsoleteAttribute is applicable to all program elements except assemblies, modules, parameters, and return values. Marking an element as obsolete informs users that the element may be removed in a future version of the product.

The string assigned to the Message property is emitted by the compiler when the attribute target is used in code. Ideally, the string should provide some workaround or programmatic alternative.

Use the IsError property to indicate to the compiler whether using the ObsoleteAttribute attribute should cause it to emit an error (IsError is `true`) or a warning (IsError is `false`).

For more information about using attributes, see Attributes.

Constructors

Expand table

ObsoleteAttribute()	Initializes a new instance of the ObsoleteAttribute class with default properties.
Obsolete Attribute(String)	Initializes a new instance of the ObsoleteAttribute class with a specified workaround message.
Obsolete Attribute(String, Boolean)	Initializes a new instance of the ObsoleteAttribute class with a workaround message and a Boolean value indicating whether the obsolete element usage is considered an error.

Properties

Expand table

Diagnostic Id	Gets or sets the ID that the compiler will use when reporting a use of the API.
IsError	Gets a value that indicates whether the compiler will treat usage of the obsolete program element as an error.
Message	Gets the workaround message.
TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)
UrlFormat	Gets or sets the URL for corresponding documentation. The API accepts a format string instead of an actual URL, creating a generic URL that includes the diagnostic ID.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance.

	(Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute
- Extend metadata using attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ObsoleteAttribute Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ObsoleteAttribute class.

Overloads

Expand table

ObsoleteAttribute()	Initializes a new instance of the ObsoleteAttribute class with default properties.
ObsoleteAttribute(String)	Initializes a new instance of the ObsoleteAttribute class with a specified workaround message.
ObsoleteAttribute(String, Boolean)	Initializes a new instance of the ObsoleteAttribute class with a workaround message and a Boolean value indicating whether the obsolete element usage is considered an error.

ObsoleteAttribute()

Initializes a new instance of the ObsoleteAttribute class with default properties.

C#

```
public ObsoleteAttribute ();
```

Remarks

The following table shows the initial property values for an instance of ObsoleteAttribute.

Expand table

Property	Value
IsError	`false`
Message	`null`
DiagnosticId	`null`
UrlFormat	`null`

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ObsoleteAttribute(String)

Initializes a new instance of the ObsoleteAttribute class with a specified workaround message.

C#

```
public ObsoleteAttribute (string? message);
```

Parameters

`message` String

The text string that describes alternative workarounds.

Remarks

The following table shows the initial property values for an instance of ObsoleteAttribute.

Expand table

Property	Value
IsError	`false`.
Message	The workaround message.
DiagnosticId	`null`.
UrlFormat	`null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ObsoleteAttribute(String, Boolean)

Initializes a new instance of the ObsoleteAttribute class with a workaround message and a Boolean value indicating whether the obsolete element usage is considered an error.

C#

```
public ObsoleteAttribute (string? message, bool error);
```

Parameters

`message` String

The text string that describes alternative workarounds.

`error` Boolean

`true` if the obsolete element usage generates a compiler error; `false` if it generates a compiler warning.

Remarks

The following table shows the initial property values for an instance of ObsoleteAttribute.

Expand table

Property	Value
IsError	The `error` value.
Message	The `message` value.
DiagnosticId	`null`.
UrlFormat	`null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ObsoleteAttribute.DiagnosticId Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the ID that the compiler will use when reporting a use of the API.

C#

```
public string? DiagnosticId { get; set; }
```

Property Value

String

The unique diagnostic ID.

Remarks

The diagnostic ID is shown in build output for warnings and errors.

This property represents the unique ID that can be used to suppress the warnings or errors, if needed.

Choose an appropriate diagnostic ID such that it is unique.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ObsoleteAttribute.IsError Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the compiler will treat usage of the obsolete program element as an error.

```
C#

public bool IsError { get; }
```

Property Value

Boolean

`true` if the obsolete element usage is considered an error; otherwise, `false`. The default is `false`.

Examples

The following example defines a class that contains two members marked as obsolete. The first, a property named `OldProperty`, produces a compiler warning if it is called. The second, a method named `CallOldMethod`, produces a compiler error. The example uses reflection to get information about the ObsoleteAttribute attributes applied to members of the type and displays the values of their Message and IsError properties.

```
C#

using System;
using System.Reflection;

public class Example
{
   // Mark OldProperty As Obsolete.
   [ObsoleteAttribute("This property is obsolete. Use NewProperty instead.",
false)]
   public string OldProperty
   { get { return "The old property value."; } }
```

```
   public string NewProperty
   { get { return "The new property value."; } }

   // Mark OldMethod As Obsolete.
   [ObsoleteAttribute("This method is obsolete. Call NewMethod instead.",
true)]
   public string OldMethod()
   {
      return "You have called OldMethod.";
   }

   public string NewMethod()
   {
      return "You have called NewMethod.";
   }

   public static void Main()
   {
      // Get all public members of this type.
      MemberInfo[] members = typeof(Example).GetMembers();
      // Count total obsolete members.
      int n = 0;

      // Try to get the ObsoleteAttribute for each public member.
      Console.WriteLine("Obsolete members in the Example class:\n");
      foreach (var member in members) {
         ObsoleteAttribute[] attribs = (ObsoleteAttribute[])

member.GetCustomAttributes(typeof(ObsoleteAttribute),
                                                                    false);
         if (attribs.Length > 0) {
            ObsoleteAttribute attrib = attribs[0];
            Console.WriteLine("Member Name: {0}.{1}",
member.DeclaringType.FullName, member.Name);
            Console.WriteLine("   Message: {0}", attrib.Message);
            Console.WriteLine("   Warning/Error: {0}", attrib.IsError ?
"Error" : "Warning");
            n++;
         }
      }

      if (n == 0)
         Console.WriteLine("The Example type has no obsolete attributes.");
   }
}
// The example displays the following output:
//       Obsolete members in the Example class:
//
//       Member Name: Example.OldMethod
//          Message: This method is obsolete. Call NewMethod instead.
//          Warning/Error: Error
//       Member Name: Example.OldProperty
//          Message: This property is obsolete. Use NewProperty instead.
//          Warning/Error: Warning
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ObsoleteAttribute.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the workaround message.

C#

```
public string? Message { get; }
```

Property Value

String

The workaround text string.

Examples

The following example defines a class that contains two members marked as obsolete. The first, a property named `OldProperty`, produces a compiler warning if it is called. The second, a method named `CallOldMethod`, produces a compiler error. The example uses reflection to get information about the ObsoleteAttribute attributes applied to members of the type and displays the values of their Message and IsError properties.

C#

```
using System;
using System.Reflection;

public class Example
{
   // Mark OldProperty As Obsolete.
   [ObsoleteAttribute("This property is obsolete. Use NewProperty instead.",
false)]
   public string OldProperty
   { get { return "The old property value."; } }

   public string NewProperty
   { get { return "The new property value."; } }
```

```
   // Mark OldMethod As Obsolete.
   [ObsoleteAttribute("This method is obsolete. Call NewMethod instead.",
true)]
   public string OldMethod()
   {
      return "You have called OldMethod.";
   }

   public string NewMethod()
   {
      return "You have called NewMethod.";
   }

   public static void Main()
   {
      // Get all public members of this type.
      MemberInfo[] members = typeof(Example).GetMembers();
      // Count total obsolete members.
      int n = 0;

      // Try to get the ObsoleteAttribute for each public member.
      Console.WriteLine("Obsolete members in the Example class:\n");
      foreach (var member in members) {
         ObsoleteAttribute[] attribs = (ObsoleteAttribute[])

member.GetCustomAttributes(typeof(ObsoleteAttribute),
                                                                false);
         if (attribs.Length > 0) {
            ObsoleteAttribute attrib = attribs[0];
            Console.WriteLine("Member Name: {0}.{1}",
member.DeclaringType.FullName, member.Name);
            Console.WriteLine("   Message: {0}", attrib.Message);
            Console.WriteLine("   Warning/Error: {0}", attrib.IsError ?
"Error" : "Warning");
            n++;
         }
      }

      if (n == 0)
         Console.WriteLine("The Example type has no obsolete attributes.");
   }
}
// The example displays the following output:
//       Obsolete members in the Example class:
//
//       Member Name: Example.OldMethod
//          Message: This method is obsolete. Call NewMethod instead.
//          Warning/Error: Error
//       Member Name: Example.OldProperty
//          Message: This property is obsolete. Use NewProperty instead.
//          Warning/Error: Warning
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ObsoleteAttribute.UrlFormat Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the URL for corresponding documentation. The API accepts a format string instead of an actual URL, creating a generic URL that includes the diagnostic ID.

C#

```
public string? UrlFormat { get; set; }
```

Property Value

String

The format string that represents a URL to corresponding documentation.

Remarks

An example format string is `https://contoso.com/obsoletion-warnings/{0}`.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

OperatingSystem Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents information about an operating system, such as the version and platform identifier. This class cannot be inherited.

C#

```
public sealed class OperatingSystem : ICloneable,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → OperatingSystem

Implements ICloneable , ISerializable

Examples

The following code example uses the OperatingSystem object to display information about the runtime operating system.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      var os = Environment.OSVersion;
      Console.WriteLine("Current OS Information:\n");
      Console.WriteLine("Platform: {0:G}", os.Platform);
      Console.WriteLine("Version String: {0}", os.VersionString);
      Console.WriteLine("Version Information:");
      Console.WriteLine("   Major: {0}", os.Version.Major);
      Console.WriteLine("   Minor: {0}", os.Version.Minor);
      Console.WriteLine("Service Pack: '{0}'", os.ServicePack);
   }
}
// If run on a Windows 8.1 system, the example displays output like the
```

```
following:
//       Current OS Information:
//
//       Platform: Win32NT
//       Version String: Microsoft Windows NT 6.2.9200.0
//       Version Information:
//          Major: 6
//          Minor: 2
//       Service Pack: ''
// If run on a Windows 7 system, the example displays output like the
following:
//       Current OS Information:
//
//       Platform: Win32NT
//       Version String: Microsoft Windows NT 6.1.7601 Service Pack 1
//       Version Information:
//          Major: 6
//          Minor: 1
//       Service Pack: 'Service Pack 1'
```

Remarks

The OperatingSystem class contains information about an operating system.

For information about the current runtime operating system, retrieve the OperatingSystem object returned by the Environment.OSVersion property. For a list of Windows operating system versions and their corresponding version numbers returned by the Version and VersionString properties and the ToString method, see Operating System Version.

By design, the OperatingSystem class is not a general purpose means of describing an operating system, and you cannot derive a more inclusive type from the OperatingSystem class. If you need a type to contain other information about an operating system, create your own type, then include a field of type OperatingSystem and any additional fields, properties, or methods that you require.

Constructors

Expand table

OperatingSystem(PlatformID, Version)	Initializes a new instance of the OperatingSystem class, using the specified platform identifier value and version object.

Properties

Expand table

Platform	Gets a PlatformID enumeration value that identifies the operating system platform.
Service Pack	Gets the service pack version represented by this OperatingSystem object.
Version	Gets a Version object that identifies the operating system.
Version String	Gets the concatenated string representation of the platform identifier, version, and service pack that are currently installed on the operating system.

Methods

Expand table

Clone()	Creates an OperatingSystem object that is identical to this instance.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo object with the data necessary to deserialize this instance.
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsAndroid()	Indicates whether the current application is running on Android.
IsAndroidVersionAtLeast(Int32, Int32, Int32, Int32)	Checks if the Android version (returned by the Linux command `uname`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified version.
IsBrowser()	Indicates whether the current application is running as WASM in a browser.
IsFreeBSD()	Indicates whether the current application is running on FreeBSD.
IsFreeBSDVersionAtLeast(Int32, Int32, Int32, Int32)	Checks if the FreeBSD version (returned by the Linux command `uname`) is greater than or equal to the specified version. This

	method can be used to guard APIs that were added in the specified version.
IsIOS()	Indicates whether the current application is running on iOS or MacCatalyst.
IsIOSVersionAtLeast(Int32, Int32, Int32)	Checks if the iOS/MacCatalyst version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified iOS version.
IsLinux()	Indicates whether the current application is running on Linux.
IsMacCatalyst()	Indicates whether the current application is running on Mac Catalyst.
IsMacCatalystVersion AtLeast(Int32, Int32, Int32)	Check for the Mac Catalyst version (iOS version as presented in Apple documentation) with a ≤ version comparison. Used to guard APIs that were added in the given Mac Catalyst release.
IsMacOS()	Indicates whether the current application is running on macOS.
IsMacOSVersionAtLeast(Int32, Int32, Int32)	Checks if the macOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified macOS version.
IsOSPlatform(String)	Indicates whether the current application is running on the specified platform.
IsOSPlatformVersion AtLeast(String, Int32, Int32, Int32, Int32)	Checks if the operating system version is greater than or equal to the specified platform version. This method can be used to guard APIs that were added in the specified OS version.
IsTvOS()	Indicates whether the current application is running on tvOS.
IsTvOSVersionAtLeast(Int32, Int32, Int32)	Checks if the tvOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified tvOS version.
IsWasi()	Indicates whether the current application is running as WASI.
IsWatchOS()	Indicates whether the current application is running on watchOS.
IsWatchOSVersion AtLeast(Int32, Int32, Int32)	Checks if the watchOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified watchOS version.
IsWindows()	Indicates whether the current application is running on Windows.

IsWindowsVersionAtLeast(Int32, Int32, Int32, Int32)	Checks if the Windows version (returned by `RtlGetVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified Windows version.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Converts the value of this OperatingSystem object to its equivalent string representation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem(PlatformID, Version) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the OperatingSystem class, using the specified platform identifier value and version object.

```
C#

public OperatingSystem (PlatformID platform, Version version);
```

Parameters

`platform` PlatformID

One of the PlatformID values that indicates the operating system platform.

`version` Version

A Version object that indicates the version of the operating system.

Exceptions

ArgumentNullException

`version` is `null`.

ArgumentException

`platform` is not a PlatformID enumeration value.

Notes to Callers

If you are using the OperatingSystem class to retrieve information about the runtime operating system, you should not call this constructor to instantiate an OperatingSystem object. Instead, you can retrieve an OperatingSystem object that describes the operating system from the OSVersion property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.Platform Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a PlatformID enumeration value that identifies the operating system platform.

C#

```
public PlatformID Platform { get; }
```

Property Value

PlatformID

One of the PlatformID values.

Examples

The following code example creates several OperatingSystem objects and displays the Platform property for each.

C#

```
// Example for the OperatingSystem.Platform and
// OperatingSystem.Version properties.
using System;

class PlatformVersionDemo
{
    // Create an OperatingSystem object and display the Platform
    // and Version properties.
    static void BuildOSObj( PlatformID pID, Version ver )
    {
        OperatingSystem opSys   = new OperatingSystem( pID, ver );
        PlatformID platform     = opSys.Platform;
        Version version         = opSys.Version;

        Console.WriteLine( "   Platform: {0,-15} Version: {1}",
            platform, version );
    }
```

```
    static void BuildOperatingSystemObjects( )
    {
        // The Version object does not need to correspond to an
        // actual OS version.
        Version verNull     = new Version( );
        Version verString   = new Version( "3.5.8.13" );
        Version verMajMin   = new Version( 6, 10 );
        Version verMMBld    = new Version( 5, 25, 5025 );
        Version verMMBVer   = new Version( 5, 6, 7, 8 );

        // All PlatformID members are shown here.
        BuildOSObj( PlatformID.Win32NT, verNull );
        BuildOSObj( PlatformID.Win32S, verString );
        BuildOSObj( PlatformID.Win32Windows, verMajMin );
        BuildOSObj( PlatformID.WinCE, verMMBld );
        BuildOSObj( PlatformID.Win32NT, verMMBVer );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of OperatingSystem.Platform " +
            "and OperatingSystem.Version \n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Create several OperatingSystem objects " +
            "and display their properties:\n" );

        BuildOperatingSystemObjects( );

        Console.WriteLine(
            "\nThe operating system of the host computer is:\n" );

        BuildOSObj(
            Environment.OSVersion.Platform,
            Environment.OSVersion.Version );
    }
}

/*
This example of OperatingSystem.Platform and OperatingSystem.Version
generates the following output.

Create several OperatingSystem objects and display their properties:

Platform: Win32NT         Version: 0.0
Platform: Win32S          Version: 3.5.8.13
Platform: Win32Windows    Version: 6.10
Platform: WinCE           Version: 5.25.5025
Platform: Win32NT         Version: 5.6.7.8

The operating system of the host computer is:
```

```
 Platform: Win32NT         Version: 5.1.2600.0
 */
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.ServicePack Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the service pack version represented by this OperatingSystem object.

C#

```
public string ServicePack { get; }
```

Property Value

String

The service pack version, if service packs are supported and at least one is installed; otherwise, an empty string ("").

Examples

The following code example demonstrates the ServicePack property.

C#

```
// This example demonstrates the OperatingSystem.ServicePack property.
using System;

class Sample
{
    public static void Main()
    {
    OperatingSystem os = Environment.OSVersion;
    String sp = os.ServicePack;
    Console.WriteLine("Service pack version = \"{0}\"", sp);
    }
}
/*
This example produces the following results:

Service pack version = "Service Pack 1"
```

```
*/
```

Remarks

A service pack is used to distribute software updates for the operating system on which it is installed. However, not every operating system uses service packs.

The service pack version, if available, is already correctly formatted for display.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Empty

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.Version Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a Version object that identifies the operating system.

C#

```
public Version Version { get; }
```

Property Value

Version

A Version object that describes the major version, minor version, build, and revision numbers for the operating system.

Examples

The following code example creates several OperatingSystem objects and displays the Version property for each.

C#

```
// Example for the OperatingSystem.Platform and
// OperatingSystem.Version properties.
using System;

class PlatformVersionDemo
{
    // Create an OperatingSystem object and display the Platform
    // and Version properties.
    static void BuildOSObj( PlatformID pID, Version ver )
    {
        OperatingSystem opSys   = new OperatingSystem( pID, ver );
        PlatformID platform     = opSys.Platform;
        Version version         = opSys.Version;

        Console.WriteLine( "   Platform: {0,-15} Version: {1}",
            platform, version );
```

```
    }

    static void BuildOperatingSystemObjects( )
    {
        // The Version object does not need to correspond to an
        // actual OS version.
        Version verNull     = new Version( );
        Version verString   = new Version( "3.5.8.13" );
        Version verMajMin   = new Version( 6, 10 );
        Version verMMBld    = new Version( 5, 25, 5025 );
        Version verMMBVer   = new Version( 5, 6, 7, 8 );

        // All PlatformID members are shown here.
        BuildOSObj( PlatformID.Win32NT, verNull );
        BuildOSObj( PlatformID.Win32S, verString );
        BuildOSObj( PlatformID.Win32Windows, verMajMin );
        BuildOSObj( PlatformID.WinCE, verMMBld );
        BuildOSObj( PlatformID.Win32NT, verMMBVer );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of OperatingSystem.Platform " +
            "and OperatingSystem.Version \n" +
            "generates the following output.\n" );
        Console.WriteLine(
            "Create several OperatingSystem objects " +
            "and display their properties:\n" );

        BuildOperatingSystemObjects( );

        Console.WriteLine(
            "\nThe operating system of the host computer is:\n" );

        BuildOSObj(
            Environment.OSVersion.Platform,
            Environment.OSVersion.Version );
    }
}

/*
This example of OperatingSystem.Platform and OperatingSystem.Version
generates the following output.

Create several OperatingSystem objects and display their properties:

Platform: Win32NT         Version: 0.0
Platform: Win32S          Version: 3.5.8.13
Platform: Win32Windows    Version: 6.10
Platform: WinCE           Version: 5.25.5025
Platform: Win32NT         Version: 5.6.7.8

The operating system of the host computer is:
```

```
Platform: Win32NT         Version: 5.1.2600.0
*/
```

Remarks

For a list of Windows operating system versions and their corresponding version numbers, see Operating System Version.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.VersionString Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the concatenated string representation of the platform identifier, version, and service pack that are currently installed on the operating system.

C#

```
public string VersionString { get; }
```

Property Value

String

The string representation of the values returned by the Platform, Version, and ServicePack properties.

Examples

The following code example demonstrates the VersionString property.

C#

```
// This example demonstrates the VersionString property.
using System;

class Sample
{
    public static void Main()
    {
    OperatingSystem os = Environment.OSVersion;
// Display the value of OperatingSystem.VersionString. By default, this is
// the same value as OperatingSystem.ToString.
    Console.WriteLine("This operating system is {0}", os.VersionString);
    }
}
/*
This example produces the following results:
```

```
This operating system is Microsoft Windows NT 5.1.2600.0 Service Pack 1
*/
```

Remarks

By default, the value returned by VersionString is the same as the value returned by the ToString method. However, an implementation of the .NET Framework for a different platform might return a more appropriate string for that platform.

For a list of Windows operating system versions and their corresponding version numbers, see Operating System Version.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ToString()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an OperatingSystem object that is identical to this instance.

C#

```
public object Clone ();
```

Returns

Object

An OperatingSystem object that is a copy of this instance.

Implements

Clone()

Examples

The following code example illustrates the use of the Clone method to make a copy of an OperatingSystem object. The clone is compared with the original object to show that they are not the same object.

C#

```
// Example for the OperatingSystem.Clone method.
using System;

class CloneCompareDemo
{
    // Copy, clone, and duplicate an OperatingSystem object.
    static void CopyOperatingSystemObjects( )
    {
        // The Version object does not need to correspond to an
        // actual OS version.
        Version verMMBVer = new Version( 5, 6, 7, 8 );
```

```
        OperatingSystem opCreate1 = new
            OperatingSystem( PlatformID.Win32NT, verMMBVer );

        // Create another OperatingSystem object with the same
        // parameters as opCreate1.
        OperatingSystem opCreate2 = new
            OperatingSystem( PlatformID.Win32NT, verMMBVer );

        // Clone opCreate1 and copy the opCreate1 reference.
        OperatingSystem opClone =
            (OperatingSystem)opCreate1.Clone( );
        OperatingSystem opCopy = opCreate1;

        // Compare the various objects for equality.
        Console.WriteLine( "{0,-50}{1}",
            "Is the second object the same as the original?",
            opCreate1.Equals( opCreate2 ) );
        Console.WriteLine( "{0,-50}{1}",
            "Is the object clone the same as the original?",
            opCreate1.Equals( opClone ) );
        Console.WriteLine( "{0,-50}{1}",
            "Is the copied object the same as the original?",
            opCreate1.Equals( opCopy ) );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of OperatingSystem.Clone( ) " +
            "generates the following output.\n" );
        Console.WriteLine(
            "Create an OperatingSystem object, and then " +
            "create another object with the \n" +
            "same parameters. Clone and copy the original " +
            "object, and then compare \n" +
            "each object with the original " +
            "using the Equals( ) method. Equals( ) \n" +
            "returns true only when both " +
            "references refer to the same object.\n" );

        CopyOperatingSystemObjects( );
    }
}

/*
This example of OperatingSystem.Clone( ) generates the following output.

Create an OperatingSystem object, and then create another object with the
same parameters. Clone and copy the original object, and then compare
each object with the original using the Equals( ) method. Equals( )
returns true only when both references refer to the same object.

Is the second object the same as the original?    False
Is the object clone the same as the original?     False
```

```
Is the copied object the same as the original?    True
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo object with the data necessary to deserialize this instance.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with serialization information.

`context` StreamingContext

The place to store and retrieve serialized data. Reserved for future use.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

The `context` parameter is reserved for future use; it is currently not implemented in the GetObjectData method.

For more information, see the SerializationInfo.AddValue method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsAndroid Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on Android.

C#

```
public static bool IsAndroid ();
```

Returns

Boolean

`true` if the current application is running on Android; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsAndroidVersionAtLeast(Int32, Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the Android version (returned by the Linux command `uname`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified version.

C#

```
public static bool IsAndroidVersionAtLeast (int major, int minor = 0, int build = 0, int revision = 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

`revision` Int32

The revision release number.

Returns

Boolean

`true` if the current application is running on an Android version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsBrowser Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running as WASM in a browser.

C#

```
public static bool IsBrowser ();
```

Returns

Boolean

`true` if the current application is running as WASM in a browser; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsFreeBSD Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on FreeBSD.

C#

```
public static bool IsFreeBSD ();
```

Returns

Boolean

`true` if the current application is running on FreeBSD; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsFreeBSDVersionAtLeast(Int32, Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the FreeBSD version (returned by the Linux command `uname`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified version.

C#

```
public static bool IsFreeBSDVersionAtLeast (int major, int minor = 0, int
build = 0, int revision = 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

`revision` Int32

The revision release number.

Returns

Boolean

`true` if the current application is running on a FreeBSD version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsIOS Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on iOS or MacCatalyst.

C#

```
[System.Runtime.Versioning.SupportedOSPlatformGuard("maccatalyst")]
public static bool IsIOS ();
```

Returns

Boolean

`true` if the current application is running on iOS or MacCatalyst; `false` otherwise.

Attributes SupportedOSPlatformGuardAttribute

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsIOSVersionAtLeast(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the iOS/MacCatalyst version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified iOS version.

C#

```
[System.Runtime.Versioning.SupportedOSPlatformGuard("maccatalyst")]
public static bool IsIOSVersionAtLeast (int major, int minor = 0, int build
= 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

Returns

Boolean

`true` if the current application is running on an iOS/MacCatalyst version that is at least what was specified in the parameters; `false` otherwise.

Attributes SupportedOSPlatformGuardAttribute

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OperatingSystem.IsLinux Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on Linux.

C#

```
public static bool IsLinux ();
```

Returns

Boolean

`true` if the current application is running on Linux; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsMacCatalyst Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on Mac Catalyst.

C#

```
public static bool IsMacCatalyst ();
```

Returns

Boolean

`true` if the current application is running on Mac Catalyst; `false` otherwise.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsMacCatalystVersionAtLeast(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Check for the Mac Catalyst version (iOS version as presented in Apple documentation) with a ≤ version comparison. Used to guard APIs that were added in the given Mac Catalyst release.

```
C#

public static bool IsMacCatalystVersionAtLeast (int major, int minor = 0,
int build = 0);
```

Parameters

`major` Int32

The version major number.

`minor` Int32

The version minor number.

`build` Int32

The version build number.

Returns

Boolean

`true` if the Mac Catalyst version is greater or equal than the specified version comparison; `false` otherwise.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OperatingSystem.IsMacOS Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on macOS.

C#

```
public static bool IsMacOS ();
```

Returns

Boolean

`true` if the current application is running on macOS; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsMacOSVersionAtLeast(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the macOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified macOS version.

```
C#

public static bool IsMacOSVersionAtLeast (int major, int minor = 0, int
build = 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

Returns

Boolean

`true` if the current application is running on an macOS version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OperatingSystem.IsOSPlatform(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on the specified platform.

```
C#
public static bool IsOSPlatform (string platform);
```

Parameters

`platform` String

The case-insensitive platform name. Examples: Browser, Linux, FreeBSD, Android, iOS, macOS, tvOS, watchOS, Windows.

Returns

Boolean

`true` if the current application is running on the specified platform; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

OperatingSystem.IsOSPlatformVersionAtLeast Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the operating system version is greater than or equal to the specified platform version. This method can be used to guard APIs that were added in the specified OS version.

```
C#

public static bool IsOSPlatformVersionAtLeast (string platform, int major,
int minor = 0, int build = 0, int revision = 0);
```

Parameters

platform String

The case-insensitive platform name. Examples: Browser, Linux, FreeBSD, Android, iOS, macOS, tvOS, watchOS, Windows.

major Int32

The major release number.

minor Int32

The minor release number (optional).

build Int32

The build release number (optional).

revision Int32

The revision release number (optional).

Returns

Boolean

`true` if the current application is running on the specified platform and is at least in the version specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsTvOS Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on tvOS.

C#

```
public static bool IsTvOS ();
```

Returns

Boolean

`true` if the current application is running on tvOS; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsTvOSVersionAtLeast(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the tvOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified tvOS version.

```
C#

public static bool IsTvOSVersionAtLeast (int major, int minor = 0, int build
= 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

Returns

Boolean

`true` if the current application is running on a tvOS version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OperatingSystem.IsWasi Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running as WASI.

C#

```
public static bool IsWasi ();
```

Returns

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsWatchOS Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on watchOS.

C#

```
public static bool IsWatchOS ();
```

Returns

Boolean

`true` if the current application is running on watchOS; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsWatchOSVersionAtLeast(Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the watchOS version (returned by `libobjc.get_operatingSystemVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified watchOS version.

```
C#

public static bool IsWatchOSVersionAtLeast (int major, int minor = 0, int
build = 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

Returns

Boolean

`true` if the current application is running on a watchOS version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OperatingSystem.IsWindows Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current application is running on Windows.

C#

```
public static bool IsWindows ();
```

Returns

Boolean

`true` if the current application is running on Windows; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.IsWindowsVersionAtLeast(Int32, Int32, Int32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if the Windows version (returned by `RtlGetVersion`) is greater than or equal to the specified version. This method can be used to guard APIs that were added in the specified Windows version.

C#

```
public static bool IsWindowsVersionAtLeast (int major, int minor = 0, int build = 0, int revision = 0);
```

Parameters

`major` Int32

The major release number.

`minor` Int32

The minor release number.

`build` Int32

The build release number.

`revision` Int32

The revision release number.

Returns

Boolean

`true` if the current application is running on a Windows version that is at least what was specified in the parameters; `false` otherwise.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperatingSystem.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this OperatingSystem object to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the values returned by the Platform, Version, and ServicePack properties.

Examples

The following code example illustrates the use of the ToString method to display OperatingSystem objects.

C#

```
// Example for the OperatingSystem constructor and the
// OperatingSystem.ToString( ) method.
using System;

class OpSysConstructDemo
{
    // Create and display an OperatingSystem object.
    static void BuildOSObj( PlatformID pID, Version ver )
    {
        OperatingSystem os = new OperatingSystem( pID, ver );

        Console.WriteLine( "   {0}", os.ToString( ) );
    }

    static void BuildOperatingSystemObjects( )
```

```
    {
        // The Version object does not need to correspond to an
        // actual OS version.
        Version verNull     = new Version( );
        Version verMajMin   = new Version( 3, 11 );
        Version verMMBld    = new Version( 5, 25, 625 );
        Version verMMBVer   = new Version( 5, 6, 7, 8 );
        Version verString   = new Version( "3.5.8.13" );

        // All PlatformID members are shown here.
        BuildOSObj( PlatformID.Win32NT, verNull );
        BuildOSObj( PlatformID.Win32S, verMajMin );
        BuildOSObj( PlatformID.Win32Windows, verMMBld );
        BuildOSObj( PlatformID.WinCE, verMMBVer );
        BuildOSObj( PlatformID.Win32NT, verString );
    }

    public static void Main( )
    {
        Console.WriteLine(
            "This example of the OperatingSystem constructor " +
            "and \nOperatingSystem.ToString( ) " +
            "generates the following output.\n" );
        Console.WriteLine(
            "Create and display several different " +
            "OperatingSystem objects:\n" );

        BuildOperatingSystemObjects( );

        Console.WriteLine(
            "\nThe OS version of the host computer is:\n\n   {0}",
            Environment.OSVersion.ToString( ) );
    }
}

/*
This example of the OperatingSystem constructor and
OperatingSystem.ToString( ) generates the following output.

Create and display several different OperatingSystem objects:

   Microsoft Windows NT 0.0
   Microsoft Win32S 3.11
   Microsoft Windows 98 5.25.625
   Microsoft Windows CE 5.6.7.8
   Microsoft Windows NT 3.5.8.13

The OS version of the host computer is:

   Microsoft Windows NT 5.1.2600.0
*/
```

Remarks

For a list of Windows operating system versions and their corresponding version numbers, see Operating System Version.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- VersionString

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperationCanceledException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown in a thread upon cancellation of an operation that the thread was executing.

C#

```
public class OperationCanceledException : SystemException
```

Inheritance Object → Exception → SystemException → OperationCanceledException

Derived System.Threading.Tasks.TaskCanceledException

Remarks

If the task that was canceled had an associated cancellation token, it is returned by the CancellationToken property, and the CancellationToken.IsCancellationRequested property of the token is set to `true`.

Constructors

Expand table

OperationCanceledException()	Initializes a new instance of the OperationCanceledException class with a system-supplied error message.
OperationCanceledException(CancellationToken)	Initializes a new instance of the OperationCanceledException class with a cancellation token.
OperationCanceledException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OperationCanceledException class with serialized data.

OperationCanceledException(String)	Initializes a new instance of the OperationCanceledException class with a specified error message.
OperationCanceledException(String, CancellationToken)	Initializes a new instance of the OperationCanceledException class with a specified error message and a cancellation token.
OperationCanceledException(String, Exception)	Initializes a new instance of the OperationCanceledException class with a specified error message and a reference to the inner exception that is the cause of this exception.
OperationCanceledException(String, Exception, CancellationToken)	Initializes a new instance of the OperationCanceledException class with a specified error message, a reference to the inner exception that is the cause of this exception, and a cancellation token.

Properties

Expand table

CancellationToken	Gets a token associated with the operation that was canceled.
Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
InnerException	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception.

	(Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperationCanceledException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the OperationCanceledException class.

Overloads

Expand table

OperationCanceledException()	Initializes a new instance of the OperationCanceledException class with a system-supplied error message.
OperationCanceledException(String)	Initializes a new instance of the OperationCanceledException class with a specified error message.
OperationCanceledException(CancellationToken)	Initializes a new instance of the OperationCanceledException class with a cancellation token.
OperationCanceledException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OperationCanceledException class with serialized data.
OperationCanceledException(String, Exception)	Initializes a new instance of the OperationCanceledException class with a specified error message and a reference to the inner exception that is the cause of this exception.
OperationCanceledException(String, CancellationToken)	Initializes a new instance of the OperationCanceledException class with a specified error message and a cancellation token.

OperationCanceledException(String, Exception, CancellationToken)	Initializes a new instance of the OperationCanceledException class with a specified error message, a reference to the inner exception that is the cause of this exception, and a cancellation token.

OperationCanceledException()

Initializes a new instance of the OperationCanceledException class with a system-supplied error message.

C#

```
public OperationCanceledException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The operation was canceled." This message takes into account the current system culture.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property	Value
InnerException	A nullreference (`Nothing` in Visual Basic).
Message	The localized error message string.
CancellationToken	A cancellation token created in the non-canceled state.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OperationCanceledException(String)

Initializes a new instance of the OperationCanceledException class with a specified error message.

C#

```
public OperationCanceledException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.
CancellationToken	A cancellation token created in the non-canceled state.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OperationCanceledException(CancellationToken)

Initializes a new instance of the OperationCanceledException class with a cancellation token.

C#

```
public OperationCanceledException (System.Threading.CancellationToken
token);
```

Parameters

`token` CancellationToken

A cancellation token associated with the operation that was canceled.

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The operation was canceled." This message takes into account the current system culture.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property	Value
InnerException	A nullreference (`Nothing` in Visual Basic).
Message	The localized error message string.
CancellationToken	`token`.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OperationCanceledException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the OperationCanceledException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected OperationCanceledException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

OperationCanceledException(String, Exception)

Initializes a new instance of the OperationCanceledException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public OperationCanceledException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property Type	Condition
InnerException	`innerException`.

Property Type	Condition
Message	`message`.
CancellationToken	A cancellation token created in the non-canceled state.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OperationCanceledException(String, CancellationToken)

Initializes a new instance of the OperationCanceledException class with a specified error message and a cancellation token.

C#

```
public OperationCanceledException (string? message,
System.Threading.CancellationToken token);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`token` CancellationToken

A cancellation token associated with the operation that was canceled.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	`message`.
CancellationToken	`token`.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OperationCanceledException(String, Exception, CancellationToken)

Initializes a new instance of the OperationCanceledException class with a specified error message, a reference to the inner exception that is the cause of this exception, and a cancellation token.

```
C#

public OperationCanceledException (string? message, Exception?
innerException, System.Threading.CancellationToken token);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

`token` CancellationToken

A cancellation token associated with the operation that was canceled.

Remarks

The content of `message` is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of OperationCanceledException.

Expand table

Property Type	Condition
InnerException	`innerException`.
Message	`message`.
CancellationToken	`token`.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OperationCanceledException.CancellationToken Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a token associated with the operation that was canceled.

C#

```
public System.Threading.CancellationToken CancellationToken { get; }
```

Property Value

CancellationToken

A token associated with the operation that was canceled, or a default token.

Remarks

If the token is associated with a canceled operation, the CancellationToken.IsCancellationRequested property of the token returns `true`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

OutOfMemoryException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is not enough memory to continue the execution of a program.

```
C#
public class OutOfMemoryException : SystemException
```

Inheritance Object → Exception → SystemException → OutOfMemoryException

Derived System.InsufficientMemoryException

Remarks

OutOfMemoryException uses the HRESULT `COR_E_OUTOFMEMORY`, which has the value 0x8007000E.

For a list of initial property values for an instance of OutOfMemoryException, see the OutOfMemoryException constructors.

> **Note**
>
> The value of the inherited **Data** property is always `null`.

An OutOfMemoryException exception has two major causes:

- You are attempting to expand a StringBuilder object beyond the length defined by its StringBuilder.MaxCapacity property.
- The common language runtime cannot allocate enough contiguous memory to successfully perform an operation. This exception can be thrown by any property assignment or method call that requires a memory allocation. For more

information on the cause of the OutOfMemoryException exception, see the blog post "Out of Memory" Does Not Refer to Physical Memory.

This type of OutOfMemoryException exception represents a catastrophic failure. If you choose to handle the exception, you should include a `catch` block that calls the Environment.FailFast method to terminate your app and add an entry to the system event log, as the following example does.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      try {
         // Outer block to handle any unexpected exceptions.
         try {
            string s = "This";
            s = s.Insert(2, "is ");

            // Throw an OutOfMemoryException exception.
            throw new OutOfMemoryException();
         }
         catch (ArgumentException) {
            Console.WriteLine("ArgumentException in String.Insert");
         }

         // Execute program logic.
      }
      catch (OutOfMemoryException e) {
         Console.WriteLine("Terminating application unexpectedly...");
         Environment.FailFast(String.Format("Out of Memory: {0}",
                                            e.Message));
      }
   }
}
// The example displays the following output:
//        Terminating application unexpectedly...
```

Some of the conditions under which the exception is thrown and the actions you can take to eliminate it include the following:

You are calling the StringBuilder.Insert method.

You are attempting to increase the length of a StringBuilder object beyond the size specified by its StringBuilder.MaxCapacity property. The following example illustrates the OutOfMemoryException exception thrown by a call to the StringBuilder.Insert(Int32,

String, Int32) method when the example tries to insert a string that would cause the object's Length property to exceed its maximum capacity.

C#

```
using System;
using System.Text;

public class Example
{
   public static void Main()
   {
      StringBuilder sb = new StringBuilder(15, 15);
      sb.Append("Substring #1 ");
      try {
         sb.Insert(0, "Substring #2 ", 1);
      }
      catch (OutOfMemoryException e) {
         Console.WriteLine("Out of Memory: {0}", e.Message);
      }
   }
}
// The example displays the following output:
//    Out of Memory: Insufficient memory to continue the execution of the
program.
```

You can do either of the following to address the error:

- Replace the call to the StringBuilder.StringBuilder(Int32, Int32) constructor with a call any other StringBuilder constructor overload. The maximum capacity of your StringBuilder object will be set to its default value, which is Int32.MaxValue.
- Call the StringBuilder.StringBuilder(Int32, Int32) constructor with a `maxCapacity` value that is large enough to accommodate any expansions to the StringBuilder object.

Your app runs as a 32-bit process.

32-bit processes can allocate a maximum of 2GB of virtual user-mode memory on 32-bit systems, and 4GB of virtual user-mode memory on 64-bit systems. This can make it more difficult for the common language runtime to allocate sufficient contiguous memory when a large allocation is needed. In contrast, 64-bit processes can allocate up to 8TB of virtual memory. To address this exception, recompile your app to target a 64-bit platform. For information on targeting specific platforms in Visual Studio, see How to: Configure Projects to Target Platforms.

Your app is leaking unmanaged resources

Although the garbage collector is able to free memory allocated to managed types, it does not manage memory allocated to unmanaged resources such as operating system handles (including handles to files, memory-mapped files, pipes, registry keys, and wait handles) and memory blocks allocated directly by Windows API calls or by calls to memory allocation functions such as `malloc`. Types that consume unmanaged resources implement the IDisposable interface.

If you are consuming a type that uses unmanaged resources, you should be sure to call its IDisposable.Dispose method when you have finished using it. (Some types also implement a `Close` method that is identical in function to a `Dispose` method.) For more information, see the Using Objects That Implement IDisposable topic.

If you have created a type that uses unmanaged resources, make sure that you have implemented the Dispose pattern and, if necessary, supplied a finalizer. For more information, see Implementing a Dispose method and Object.Finalize.

You are attempting to create a large array in a 64-bit process

By default, the common language runtime in .NET Framework does not allow single objects whose size exceeds 2GB. To override this default, you can use the <gcAllowVeryLargeObjects> configuration file setting to enable arrays whose total size exceeds 2 GB. On .NET Core, support for arrays of greater than 2 GB is enabled by default.

You are working with very large sets of data (such as arrays, collections, or database data sets) in memory.

When data structures or data sets that reside in memory become so large that the common language runtime is unable to allocate enough contiguous memory for them, an OutOfMemoryException exception results.

To prevent the OutOfMemoryException exceptions, you must modify your application so that less data is resident in memory, or the data is divided into segments that require smaller memory allocations. For example:

- If you are retrieving all of the data from a database and then filtering it in your app to minimize trips to the server, you should modify your queries to return only the subset of data that your app needs. When working with large tables, multiple queries are almost always more efficient than retrieving all of the data in a single table and then manipulating it.

- If you are executing queries that users create dynamically, you should ensure that the number of records returned by the query is limited.

- If you are using large arrays or other collection objects whose size results in an OutOfMemoryException exception, you should modify your application to work the data in subsets rather than to work with it all at once.

The following example gets a array that consists of 200 million floating-point values and then calculates their mean. The output from the example shows that, because the example stores the entire array in memory before it calculates the mean, an OutOfMemoryException is thrown.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Double[] values = GetData();
      // Compute mean.
      Console.WriteLine("Sample mean: {0}, N = {1}",
                        GetMean(values), values.Length);
   }

   private static Double[] GetData()
   {
      Random rnd = new Random();
      List<Double> values = new List<Double>();
      for (int ctr = 1; ctr <= 200000000; ctr++) {
         values.Add(rnd.NextDouble());
         if (ctr % 10000000 == 0)
            Console.WriteLine("Retrieved {0:N0} items of data.",
                              ctr);
      }
      return values.ToArray();
   }

   private static Double GetMean(Double[] values)
   {
      Double sum = 0;
      foreach (var value in values)
         sum += value;

      return sum / values.Length;
   }
}
// The example displays output like the following:
//    Retrieved 10,000,000 items of data.
//    Retrieved 20,000,000 items of data.
//    Retrieved 30,000,000 items of data.
//    Retrieved 40,000,000 items of data.
//    Retrieved 50,000,000 items of data.
```

```
//    Retrieved 60,000,000 items of data.
//    Retrieved 70,000,000 items of data.
//    Retrieved 80,000,000 items of data.
//    Retrieved 90,000,000 items of data.
//    Retrieved 100,000,000 items of data.
//    Retrieved 110,000,000 items of data.
//    Retrieved 120,000,000 items of data.
//    Retrieved 130,000,000 items of data.
//
//    Unhandled Exception: OutOfMemoryException.
```

The following example eliminates the OutOfMemoryException exception by processing the incoming data without storing the entire data set in memory, serializing the data to a file if necessary to permit further processing (these lines are commented out in the example, since in this case they produce a file whose size is greater than 1GB), and returning the calculated mean and the number of cases to the calling routine.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      Tuple<Double, long> result = GetResult();
      Console.WriteLine("Sample mean: {0}, N = {1:N0}",
                        result.Item1, result.Item2);
   }

   private static Tuple<Double, long> GetResult()
   {
      int chunkSize = 50000000;
      int nToGet = 200000000;
      Random rnd = new Random();
      // FileStream fs = new FileStream(@".\data.bin", FileMode.Create);
      // BinaryWriter bin = new BinaryWriter(fs);
      // bin.Write((int)0);
      int n = 0;
      Double sum = 0;
      for (int outer = 0;
           outer <= ((int) Math.Ceiling(nToGet * 1.0 / chunkSize) - 1);
           outer++) {
         for (int inner = 0;
              inner <= Math.Min(nToGet - n - 1, chunkSize - 1);
              inner++) {
            Double value = rnd.NextDouble();
            sum += value;
            n++;
            // bin.Write(value);
         }
```

```
        }
        // bin.Seek(0, SeekOrigin.Begin);
        // bin.Write(n);
        // bin.Close();
        return new Tuple<Double, long>(sum/n, n);
    }
}
// The example displays output like the following:
//    Sample mean: 0.500022771458399, N = 200,000,000
```

You are repeatedly concatenating large strings.

Because strings are immutable, each string concatenation operation creates a new string. The impact for small strings, or for a small number of concatenation operations, is negligible. But for large strings or a very large number of concatenation operations, string concatenation can lead to a large number of memory allocations and memory fragmentation, poor performance, and possibly OutOfMemoryException exceptions.

When concatenating large strings or performing a large number of concatenation operations, you should use the StringBuilder class instead of the String class. When you have finished manipulating the string, convert the StringBuilder instance to a string by calling the StringBuilder.ToString method.

You pin a large number of objects in memory.

Pinning a large number of objects in memory for long periods can make it difficult for the garbage collector to allocate contiguous blocks of memory. If you've pinned a large number of objects in memory, for example by using the `fixed` statement in C# or by calling the GCHandle.Alloc(Object, GCHandleType) method with a handle type of GCHandleType.Pinned, you can do the following to address the OutOfMemoryException exception.

- Evaluate whether each object really needs to be pinned,
- Ensure that each object is unpinned as soon as possible.
- Make sure that each call to the GCHandle.Alloc(Object, GCHandleType) method to pin memory has a corresponding call to the GCHandle.Free method to unpin that memory.

The following Microsoft intermediate (MSIL) instructions throw an OutOfMemoryException exception:

- box
- newarr
- newobj

Constructors

Expand table

OutOfMemoryException()	Initializes a new instance of the OutOfMemoryException class.
OutOfMemoryException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OutOfMemoryException class with serialized data.
OutOfMemoryException(String)	Initializes a new instance of the OutOfMemoryException class with a specified error message.
OutOfMemoryException(String, Exception)	Initializes a new instance of the OutOfMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OutOfMemoryException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the OutOfMemoryException class.

Overloads

Expand table

OutOfMemoryException()	Initializes a new instance of the OutOfMemoryException class.
OutOfMemoryException(String)	Initializes a new instance of the OutOfMemoryException class with a specified error message.
OutOfMemoryException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OutOfMemoryException class with serialized data.
OutOfMemoryException(String, Exception)	Initializes a new instance of the OutOfMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

OutOfMemoryException()

Initializes a new instance of the OutOfMemoryException class.

C#

```
public OutOfMemoryException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "There was not enough memory to continue the execution of the program." This message takes into account the current system culture.

The following table shows the initial property values for an instance of OutOfMemoryException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

OutOfMemoryException(String)

Initializes a new instance of the OutOfMemoryException class with a specified error message.

C#

```
public OutOfMemoryException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of OutOfMemoryException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

OutOfMemoryException(SerializationInfo, StreamingContext)

⊗ Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the OutOfMemoryException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected OutOfMemoryException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

OutOfMemoryException(String, Exception)

Initializes a new instance of the OutOfMemoryException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public OutOfMemoryException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of OutOfMemoryException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OverflowException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.

C#

```
public class OverflowException : ArithmeticException
```

Inheritance Object → Exception → SystemException → ArithmeticException → OverflowException

Remarks

An OverflowException is thrown at run time under the following conditions:

- An arithmetic operation produces a result that is outside the range of the data type returned by the operation. The following example illustrates the OverflowException that is thrown by a multiplication operation that overflows the bounds of the Int32 type.

  C#

  ```
  int value = 780000000;
  checked {
  try {
     // Square the original value.
     int square = value * value;
     Console.WriteLine("{0} ^ 2 = {1}", value, square);
  }
  catch (OverflowException) {
     double square = Math.Pow(value, 2);
     Console.WriteLine("Exception: {0} > {1:E}.",
                       square, Int32.MaxValue);
  } }
  ```

```
// The example displays the following output:
//       Exception: 6.084E+17 > 2.147484E+009.
```

- A casting or conversion operation attempts to perform a narrowing conversion, and the value of the source data type is outside the range of the target data type. The following example illustrates the OverflowException that is thrown by the attempt to convert a large unsigned byte value to a signed byte value.

```C#
byte value = 241;
checked {
try {
   sbyte newValue = (sbyte) value;
   Console.WriteLine("Converted the {0} value {1} to the {2} value
{3}.",
                     value.GetType().Name, value,
                     newValue.GetType().Name, newValue);
}
catch (OverflowException) {
   Console.WriteLine("Exception: {0} > {1}.", value, SByte.MaxValue);
} }
// The example displays the following output:
//       Exception: 241 > 127.
```

In each case, the result of the operation is a value that is less than the `MinValue` property or greater than the `MaxValue` property of the data type that results from the operation.

For the arithmetic, casting, or conversion operation to throw an OverflowException, the operation must occur in a checked context. By default, arithmetic operations and overflows in Visual Basic are checked; in C# and F#, they are not. If the operation occurs in an unchecked context, the result is truncated by discarding any high-order bits that do not fit into the destination type. The following example illustrates such an unchecked conversion in C# or F#. It repeats the previous example in an unchecked context.

```C#
byte value = 241;
try {
   sbyte newValue = (sbyte) value;
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                     value.GetType().Name, value,
                     newValue.GetType().Name, newValue);
}
catch (OverflowException) {
   Console.WriteLine("Exception: {0} > {1}.", value, SByte.MaxValue);
}
```

```
// The example displays the following output:
//       Converted the Byte value 241 to the SByte value -15.
```

The following Microsoft intermediate language (MSIL) instructions throw an OverflowException:

- `add.ovf.` *<signed>*
- `conv.ovf.` *<to type>*
- `conv.ovf.` *<to type>* `.un`
- `mul.ovf.` *<type>*
- `sub.ovf.` *<type>*
- `newarr`

OverflowException uses the HRESULT COR_E_OVERFLOW, which has the value 0x80131516.

For a list of initial property values for an instance of OverflowException, see the OverflowException constructors.

Constructors

Expand table

OverflowException()	Initializes a new instance of the OverflowException class.
OverflowException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OverflowException class with serialized data.
OverflowException(String)	Initializes a new instance of the OverflowException class with a specified error message.
OverflowException(String, Exception)	Initializes a new instance of the OverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)

GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

OverflowException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the OverflowException class.

Overloads

Expand table

OverflowException()	Initializes a new instance of the OverflowException class.
OverflowException(String)	Initializes a new instance of the OverflowException class with a specified error message.
OverflowException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the OverflowException class with serialized data.
OverflowException(String, Exception)	Initializes a new instance of the OverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

OverflowException()

Initializes a new instance of the OverflowException class.

C#

```
public OverflowException ();
```

Remarks

The following table shows the initial property values for an instance of OverflowException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

OverflowException(String)

Initializes a new instance of the OverflowException class with a specified error message.

C#

```
public OverflowException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of OverflowException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

OverflowException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the OverflowException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
```

```
protected OverflowException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

OverflowException(String, Exception)

Initializes a new instance of the OverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public OverflowException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of OverflowException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParamArrayAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that a method will allow a variable number of arguments in its invocation. This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.Parameter,
AllowMultiple=false, Inherited=true)]
public sealed class ParamArrayAttribute : Attribute
```

Inheritance Object → Attribute → ParamArrayAttribute

Attributes AttributeUsageAttribute

Examples

The following example defines a `Temperature` class that includes a `Display` method, which is intended to display one or more formatted temperature values. The method has a single parameter, `formats`, which is defined as a parameter array.

C#

```
using System;

public class Temperature
{
   private decimal temp;

   public Temperature(decimal temperature)
   {
      this.temp = temperature;
   }

   public override string ToString()
   {
      return ToString("C");
```

```
    }

    public string ToString(string format)
    {
       if (String.IsNullOrEmpty(format))
          format = "G";

       switch (format.ToUpper())
       {
          case "G":
          case "C":
             return temp.ToString("N") + "  °C";
          case "F":
             return (9 * temp / 5 + 32).ToString("N") + "  °F";
          case "K":
             return (temp + 273.15m).ToString("N") + "  °K";
          default:
             throw new FormatException(String.Format("The '{0}' format
specifier is not supported",
                                                      format));
       }
    }

    public void Display(params string []formats)
    {
       if (formats.Length == 0)
       {
          Console.WriteLine(this.ToString("G"));
       }
       else
       {
          foreach (string format in formats)
          {
             try {
                Console.WriteLine(this.ToString(format));
             }
             // If there is an exception, do nothing.
             catch { }
          }
       }
    }
}
```

The following example illustrates three different calls to the `Temperature.Display` method. In the first, the method is passed an array of format strings. In the second, the method is passed four individual format strings as arguments. In the third, the method is called with no arguments. As the output from the example illustrates, the Visual Basic and C# compilers translate this into a call to the `Display` method with an empty string array.

```
C#
```

```
public class Class1
{
   public static void Main()
   {
      Temperature temp1 = new Temperature(100);
      string[] formats = { "C", "G", "F", "K" };

      // Call Display method with a string array.
      Console.WriteLine("Calling Display with a string array:");
      temp1.Display(formats);
      Console.WriteLine();

      // Call Display method with individual string arguments.
      Console.WriteLine("Calling Display with individual arguments:");
      temp1.Display("C", "F", "K", "G");
      Console.WriteLine();

      // Call parameterless Display method.
      Console.WriteLine("Calling Display with an implicit parameter
array:");
      temp1.Display();
   }
}
// The example displays the following output:
//       Calling Display with a string array:
//       100.00  °C
//       100.00  °C
//       212.00  °F
//       373.15  °K
//
//       Calling Display with individual arguments:
//       100.00  °C
//       212.00  °F
//       373.15  °K
//       100.00  °C
//
//       Calling Display with an implicit parameter array:
//       100.00  °C
```

Remarks

The ParamArrayAttribute indicates that a method parameter is a parameter array. A parameter array allows the specification of an unknown number of arguments. A parameter array must be the last parameter in a formal parameter list, and it must be a single-dimension array. When the method is called, a parameter array permits arguments to a method to be specified in either of two ways:

- As a single expression of a type that is implicitly convertible to the parameter array type. The parameter array functions as a value parameter.

- As zero or more arguments, where each argument is an expression of a type that is implicitly convertible to the type of the parameter array element.

The example in the next section illustrates both calling conventions.

> **Note**
>
> Typically, the **ParamArrayAttribute** is not used directly in code. Instead, individual language keywords, such as `ParamArray` in Visual Basic and `params` in C#, are used as wrappers for the **ParamArrayAttribute** class. Some languages, such as C#, may even require the use of the language keyword and prohibit the use of **ParamArrayAttribute**.

During overload resolution, when compilers that support parameter arrays encounter a method overload that does not exist but has one fewer parameter than an overload that includes a parameter array, they will replace the method with the overload that includes the parameter array. For example, a call to the `String.Split()` instance method (which does not exist in the String class) is resolved as a call to the String.Split(Char[]) method. The compiler will also pass an empty array of the required type to the method. This means that the method must always be prepared to handle an array whose length is zero when it processes the elements in the parameter array. The example provides an illustration.

For more information about using attributes, see Attributes.

Constructors

Expand table

ParamArrayAttribute()	Initializes a new instance of the ParamArrayAttribute class with default properties.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute
- Extending Metadata Using Attributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParamArrayAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ParamArrayAttribute class with default properties.

C#

```
public ParamArrayAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

PlatformID Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Identifies the operating system, or platform, supported by an assembly.

C#

```
public enum PlatformID
```

Inheritance Object → ValueType → Enum → PlatformID

Fields

Expand table

MacOSX	6	The operating system is Macintosh. This value was returned by Silverlight. On .NET Core, its replacement is `Unix`.
Other	7	Any other operating system. This includes Browser (WASM).
Unix	4	The operating system is Unix.
Win32NT	2	The operating system is Windows NT or later.
Win32S	0	The operating system is Win32s. This value is no longer in use.
Win32Windows	1	The operating system is Windows 95 or Windows 98. This value is no longer in use.
WinCE	3	The operating system is Windows CE. This value is no longer in use.
Xbox	5	The development platform is Xbox 360. This value is no longer in use.

Examples

The following example demonstrates using the `PlatformID` class to identify the currently executing operating system:

C#

```
// This example demonstrates the PlatformID enumeration.
using System;

class Sample
{
    public static void Main()
    {
    string msg1 = "This is a Windows operating system.";
    string msg2 = "This is a Unix operating system.";
    string msg3 = "ERROR: This platform identifier is invalid.";

// Assume this example is run on a Windows operating system.

    OperatingSystem os = Environment.OSVersion;
    PlatformID     pid = os.Platform;
    switch (pid)
        {
        case PlatformID.Win32NT:
        case PlatformID.Win32S:
        case PlatformID.Win32Windows:
        case PlatformID.WinCE:
            Console.WriteLine(msg1);
            break;
        case PlatformID.Unix:
            Console.WriteLine(msg2);
            break;
        default:
            Console.WriteLine(msg3);
            break;
        }
    }
}
/*
This example produces the following results:

This is a Windows operating system.
*/
```

Remarks

Use the Environment.OSVersion and OperatingSystem.Platform properties to obtain the `PlatformID` enumeration for the currently executing operating system or development platform. Use the `PlatformID` enumeration to help determine whether the current operating system or development platform supports your application.

You can use the underlying integer value of each `PlatformID` enumeration member as the `PlatformId` argument for the SignTool.exe (Sign Tool) utility.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

PlatformNotSupportedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a feature does not run on a particular platform.

C#

```
public class PlatformNotSupportedException : NotSupportedException
```

Inheritance Object → Exception → SystemException → NotSupportedException → PlatformNotSupportedException

Remarks

`PlatformNotSupportedException` uses the HRESULT COR_E_PLATFORMNOTSUPPORTED, which has the value 0x80131539.

Constructors

Expand table

PlatformNotSupportedException()	Initializes a new instance of the PlatformNotSupportedException class with default properties.
PlatformNotSupportedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the PlatformNotSupportedException class with serialized data.
PlatformNotSupportedException(String)	Initializes a new instance of the PlatformNotSupportedException class with a specified error message.
PlatformNotSupportedException(String, Exception)	Initializes a new instance of the PlatformNotSupportedException class with a specified

	error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function.

	(Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

PlatformNotSupportedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the PlatformNotSupportedException class.

Overloads

Expand table

PlatformNotSupportedException()	Initializes a new instance of the PlatformNotSupportedException class with default properties.
PlatformNotSupportedException(String)	Initializes a new instance of the PlatformNotSupportedException class with a specified error message.
PlatformNotSupportedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the PlatformNotSupportedException class with serialized data.
PlatformNotSupportedException(String, Exception)	Initializes a new instance of the PlatformNotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

PlatformNotSupportedException()

Initializes a new instance of the PlatformNotSupportedException class with default properties.

C#

```
public PlatformNotSupportedException ();
```

Remarks

When an instance of the PlatformNotSupportedException class is created by a call to this constructor, the following properties are initialized to the specified values:

Expand table

Property	Value
InnerException	`null`.
Message	The empty string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

PlatformNotSupportedException(String)

Initializes a new instance of the PlatformNotSupportedException class with a specified error message.

C#

```
public PlatformNotSupportedException (string? message);
```

Parameters

`message` String

The text message that explains the reason for the exception.

Remarks

When an instance of the PlatformNotSupportedException class is created by a call to this constructor, the following properties are initialized to the specified values:

Expand table

Property	Value
InnerException	`null`.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

PlatformNotSupportedException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the PlatformNotSupportedException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected PlatformNotSupportedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

PlatformNotSupportedException(String, Exception)

Initializes a new instance of the PlatformNotSupportedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public PlatformNotSupportedException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of PlatformNotSupportedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Predicate<T> Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that defines a set of criteria and determines whether the specified object meets those criteria.

```
C#
public delegate bool Predicate<in T>(T obj);
```

Type Parameters

T

The type of the object to compare.
This type parameter is contravariant. That is, you can use either the type you specified or any type that is less derived. For more information about covariance and contravariance, see Covariance and Contravariance in Generics.

Parameters

obj T

The object to compare against the criteria defined within the method represented by this delegate.

Return Value

Boolean

`true` if `obj` meets the criteria defined within the method represented by this delegate; otherwise, `false`.

Examples

The following code example uses a Predicate<T> delegate with the Array.Find method to search an array of Point structures. The example explicitly defines a Predicate<T>

delegate named `predicate` and assigns it a method named `FindPoints` that returns `true` if the product of the Point.X and Point.Y fields is greater than 100,000. Note that it is customary to use a lambda expression rather than to explicitly define a delegate of type Predicate<T>, as the second example illustrates.

C#

```
using System;
using System.Drawing;

public class Example
{
   public static void Main()
   {
      // Create an array of Point structures.
      Point[] points = { new Point(100, 200),
                         new Point(150, 250), new Point(250, 375),
                         new Point(275, 395), new Point(295, 450) };

      // Define the Predicate<T> delegate.
      Predicate<Point> predicate = FindPoints;

      // Find the first Point structure for which X times Y
      // is greater than 100000.
      Point first = Array.Find(points, predicate);

      // Display the first structure found.
      Console.WriteLine("Found: X = {0}, Y = {1}", first.X, first.Y);
   }

   private static bool FindPoints(Point obj)
   {
      return obj.X * obj.Y > 100000;
   }
}
// The example displays the following output:
//        Found: X = 275, Y = 395
```

The following example is identical to the previous example, except that it uses a lambda expression to represent the Predicate<T> delegate. Each element of the `points` array is passed to the lambda expression until the expression finds an element that meets the search criteria. In this case, the lambda expression returns `true` if the product of the X and Y fields is greater than 100,000.

C#

```
using System;
using System.Drawing;

public class Example
```

```
{
   public static void Main()
   {
      // Create an array of Point structures.
      Point[] points = { new Point(100, 200),
                         new Point(150, 250), new Point(250, 375),
                         new Point(275, 395), new Point(295, 450) };

      // Find the first Point structure for which X times Y
      // is greater than 100000.
      Point first = Array.Find(points, x => x.X * x.Y > 100000 );

      // Display the first structure found.
      Console.WriteLine("Found: X = {0}, Y = {1}", first.X, first.Y);
   }
}
// The example displays the following output:
//        Found: X = 275, Y = 395
```

Remarks

This delegate is used by several methods of the Array and List<T> classes to search for elements in the collection.

Typically, the Predicate<T> delegate is represented by a lambda expression. Because locally scoped variables are available to the lambda expression, it is easy to test for a condition that is not precisely known at compile time. This is simulated in the following example, which defines a `HockeyTeam` class that contains information about a National Hockey League team and the year in which it was founded. The example defines an array of integer values that represent years, and randomly assigns one element of the array to `foundedBeforeYear`, which is a variable that is locally scoped to the example's `Main` method. Because locally scoped variables are available to a lambda expression, the lambda expression passed to the List<T>.FindAll method is able to return a `HockeyTeam` object for each team founded on or before that year.

C#

```
using System;
using System.Collections.Generic;

public class HockeyTeam
{
   private string _name;
   private int _founded;

   public HockeyTeam(string name, int year)
   {
```

```
        _name = name;
        _founded = year;
    }

    public string Name {
        get { return _name; }
    }

    public int Founded {
        get { return _founded; }
    }
}

public class Example
{
    public static void Main()
    {
        Random rnd = new Random();
        List<HockeyTeam> teams = new List<HockeyTeam>();
        teams.AddRange( new HockeyTeam[] { new HockeyTeam("Detroit Red Wings",
1926),
                                           new HockeyTeam("Chicago
Blackhawks", 1926),
                                           new HockeyTeam("San Jose Sharks",
1991),
                                           new HockeyTeam("Montreal
Canadiens", 1909),
                                           new HockeyTeam("St. Louis Blues",
1967) } );
        int[] years = { 1920, 1930, 1980, 2000 };
        int foundedBeforeYear = years[rnd.Next(0, years.Length)];
        Console.WriteLine("Teams founded before {0}:", foundedBeforeYear);
        foreach (var team in teams.FindAll( x => x.Founded <=
foundedBeforeYear))
            Console.WriteLine("{0}: {1}", team.Name, team.Founded);
    }
}
// The example displays output similar to the following:
//       Teams founded before 1930:
//       Detroit Red Wings: 1926
//       Chicago Blackhawks: 1926
//       Montreal Canadiens: 1909
```

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Find<T>(T[], Predicate<T>)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Progress<T> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an IProgress<T> that invokes callbacks for each reported progress value.

C#

```
public class Progress<T> : IProgress<T>
```

Type Parameters

T

Specifies the type of the progress report value.

Inheritance Object → Progress<T>

Implements IProgress<T>

Remarks

Any handler provided to the constructor or event handlers registered with the ProgressChanged event are invoked through a SynchronizationContext instance captured when the instance is constructed. If there is no current SynchronizationContext at the time of construction, the callbacks will be invoked on the ThreadPool.

For more information and a code example, see the article Async in 4.5: Enabling Progress and Cancellation in Async APIs in the .NET Framework blog.

Constructors

Expand table

Progress<T>()	Initializes the Progress<T> object.

Progress<T>(Action<T>)	Initializes the Progress<T> object with the specified callback.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnReport(T)	Reports a progress change.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Events

Expand table

ProgressChanged	Raised for each reported progress value.

Explicit Interface Implementations

Expand table

IProgress<T>.Report(T)	Reports a progress change.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Progress<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes the Progress<T> object or initializes the Progress<T> object with the specified callback.

Overloads

Expand table

Progress<T>()	Initializes the Progress<T> object.
Progress<T>(Action<T>)	Initializes the Progress<T> object with the specified callback.

Progress<T>()

Initializes the Progress<T> object.

C#

```
public Progress ();
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Progress<T>(Action<T>)

Initializes the Progress<T> object with the specified callback.

```
C#
public Progress (Action<T> handler);
```

Parameters

`handler` Action<T>

A handler to invoke for each reported progress value. This handler will be invoked in addition to any delegates registered with the ProgressChanged event. Depending on the SynchronizationContext instance captured by the Progress<T> at construction, it is possible that this handler instance could be invoked concurrently with itself.

Exceptions

ArgumentNullException

`handler` is `null` (`Nothing` in Visual Basic).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Progress<T>.OnReport(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports a progress change.

C#

```
protected virtual void OnReport (T value);
```

Parameters

`value` T

The value of the updated progress.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Progress<T>.ProgressChanged Event

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Raised for each reported progress value.

C#

```
public event EventHandler<T>? ProgressChanged;
```

Event Type

EventHandler<T>

Remarks

Handlers registered with this event will be invoked on the SynchronizationContext captured when the instance was constructed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Progress<T>.IProgress<T>.Report(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports a progress change.

```
C#
void IProgress<T>.Report (T value);
```

Parameters

`value` T

The value of the updated progress.

Implements

Report(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Random Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a pseudo-random number generator, which is an algorithm that produces a sequence of numbers that meet certain statistical requirements for randomness.

C#

```
public class Random
```

Inheritance Object → Random

Examples

The following example creates a single random number generator and calls its NextBytes, Next, and NextDouble methods to generate sequences of random numbers within different ranges.

C#

```
// Instantiate random number generator using system-supplied value as seed.
var rand = new Random();

// Generate and display 5 random byte (integer) values.
var bytes = new byte[5];
rand.NextBytes(bytes);
Console.WriteLine("Five random byte values:");
foreach (byte byteValue in bytes)
    Console.Write("{0, 5}", byteValue);
Console.WriteLine();

// Generate and display 5 random integers.
Console.WriteLine("Five random integer values:");
for (int ctr = 0; ctr <= 4; ctr++)
    Console.Write("{0,15:N0}", rand.Next());
Console.WriteLine();

// Generate and display 5 random integers between 0 and 100.
Console.WriteLine("Five random integers between 0 and 100:");
```

```
for (int ctr = 0; ctr <= 4; ctr++)
    Console.Write("{0,8:N0}", rand.Next(101));
Console.WriteLine();

// Generate and display 5 random integers from 50 to 100.
Console.WriteLine("Five random integers between 50 and 100:");
for (int ctr = 0; ctr <= 4; ctr++)
    Console.Write("{0,8:N0}", rand.Next(50, 101));
Console.WriteLine();

// Generate and display 5 random floating point values from 0 to 1.
Console.WriteLine("Five Doubles.");
for (int ctr = 0; ctr <= 4; ctr++)
    Console.Write("{0,8:N3}", rand.NextDouble());
Console.WriteLine();

// Generate and display 5 random floating point values from 0 to 5.
Console.WriteLine("Five Doubles between 0 and 5.");
for (int ctr = 0; ctr <= 4; ctr++)
    Console.Write("{0,8:N3}", rand.NextDouble() * 5);

// The example displays output like the following:
//    Five random byte values:
//      194  185  239   54  116
//    Five random integer values:
//        507,353,531  1,509,532,693  2,125,074,958  1,409,512,757
652,767,128
//    Five random integers between 0 and 100:
//          16      78      94      79      52
//    Five random integers between 50 and 100:
//          56      66      96      60      65
//    Five Doubles.
//       0.943   0.108   0.744   0.563   0.415
//    Five Doubles between 0 and 5.
//       2.934   3.130   0.292   1.432   4.369
```

The following example generates a random integer that it uses as an index to retrieve a string value from an array.

C#

```
Random rnd = new Random();
string[] malePetNames = { "Rufus", "Bear", "Dakota", "Fido",
                          "Vanya", "Samuel", "Koani", "Volodya",
                          "Prince", "Yiska" };
string[] femalePetNames = { "Maggie", "Penny", "Saya", "Princess",
                            "Abby", "Laila", "Sadie", "Olivia",
                            "Starlight", "Talla" };

// Generate random indexes for pet names.
int mIndex = rnd.Next(malePetNames.Length);
int fIndex = rnd.Next(femalePetNames.Length);
```

```
// Display the result.
Console.WriteLine("Suggested pet name of the day: ");
Console.WriteLine("   For a male:     {0}", malePetNames[mIndex]);
Console.WriteLine("   For a female:   {0}", femalePetNames[fIndex]);

// The example displays output similar to the following:
//       Suggested pet name of the day:
//          For a male:     Koani
//          For a female:   Maggie
```

Remarks

For more information about this API, see Supplemental API remarks for Random.

Notes to Inheritors

In .NET Framework 2.0 and later, the behavior of the Next(), Next(Int32, Int32), and NextBytes(Byte[]) methods have changed so that these methods do not necessarily call the derived class implementation of the Sample() method. As a result, classes derived from Random that target .NET Framework 2.0 and later should also override these three methods.

Notes to Callers

The implementation of the random number generator in the Random class isn't guaranteed to remain the same across major versions of .NET. As a result, you shouldn't assume that the same seed will result in the same pseudo-random sequence in different versions of .NET.

Constructors

Expand table

Random()	Initializes a new instance of the Random class using a default seed value.
Random(Int32)	Initializes a new instance of the Random class, using the specified seed value.

Properties

Expand table

Shared	Provides a thread-safe Random instance that may be used concurrently from any thread.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetItems<T>(ReadOnlySpan<T>, Int32)	Creates an array populated with items chosen at random from the provided set of choices.
GetItems<T>(ReadOnlySpan<T>, Span<T>)	Fills the elements of a specified span with items chosen at random from the provided set of choices.
GetItems<T>(T[], Int32)	Creates an array populated with items chosen at random from the provided set of choices.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Next()	Returns a non-negative random integer.
Next(Int32)	Returns a non-negative random integer that is less than the specified maximum.
Next(Int32, Int32)	Returns a random integer that is within a specified range.
NextBytes(Byte[])	Fills the elements of a specified array of bytes with random numbers.
NextBytes(Span<Byte>)	Fills the elements of a specified span of bytes with random numbers.
NextDouble()	Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0.
NextInt64()	Returns a non-negative random integer.
NextInt64(Int64)	Returns a non-negative random integer that is less than the specified maximum.

NextInt64(Int64, Int64)	Returns a random integer that is within a specified range.
NextSingle()	Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0.
Sample()	Returns a random floating-point number between 0.0 and 1.0.
Shuffle<T>(Span<T>)	Performs an in-place shuffle of a span.
Shuffle<T>(T[])	Performs an in-place shuffle of an array.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Random class.

Overloads

Expand table

Random()	Initializes a new instance of the Random class using a default seed value.
Random(Int32)	Initializes a new instance of the Random class, using the specified seed value.

Random()

Initializes a new instance of the Random class using a default seed value.

C#

```
public Random ();
```

Examples

The following example uses the parameterless constructor to instantiate three Random objects and displays a sequence of five random integers for each. If it is run on .NET Framework, because the first two Random objects are created in close succession, they are instantiated using identical seed values based on the system clock and, therefore, they produce an identical sequence of random numbers. On the other hand, the parameterless constructor of the third Random object is called after a two-second delay caused by calling the Thread.Sleep method. Because this produces a different seed value for the third Random object, it produces a different sequence of random numbers.

C#

```
using System;
using System.Threading;

public class RandomNumbers
{
   public static void Main()
   {
      Random rand1 = new Random();
      Random rand2 = new Random();
      Thread.Sleep(2000);
      Random rand3 = new Random();
      ShowRandomNumbers(rand1);
      ShowRandomNumbers(rand2);
      ShowRandomNumbers(rand3);
   }

   private static void ShowRandomNumbers(Random rand)
   {
      Console.WriteLine();
      byte[] values = new byte[5];
      rand.NextBytes(values);
      foreach (byte value in values)
         Console.Write("{0, 5}", value);
      Console.WriteLine();
   }
}
// The example displays an output similar to the following:
//       28   35  133  224   58
//
//       28   35  133  224   58
//
//       32  222   43  251   49
```

Remarks

In .NET Framework, the default seed value is derived from the system clock, which has finite resolution. As a result, different Random objects that are created in close succession by a call to the parameterless constructor have identical default seed values and, therefore, produce identical sets of random numbers. You can avoid this problem by using a single Random object to generate all random numbers. You can also work around it by generating your own random seed value and passing it to the Random(Int32) constructor. For more information, see the Random(Int32) constructor.

In .NET Core, the default seed value is produced by the thread-static, pseudo-random number generator, so the previously described limitation does not apply.

Different Random objects created in close succession produce different sets of random numbers in .NET Core.

Call this constructor if you want your random number generator to generate a random sequence of numbers. To generate a fixed sequence of random numbers that will be the same for different random number generators, call the Random(Int32) constructor with a fixed seed value. This Random constructor overload is frequently used when testing apps that use random numbers.

Once you've instantiated the random number generator, you call individual Random methods, such as Next() or NextDouble(), to generate random numbers.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Random(Int32)

Initializes a new instance of the Random class, using the specified seed value.

C#

```
public Random (int Seed);
```

Parameters

Seed Int32

A number used to calculate a starting value for the pseudo-random number sequence. If a negative number is specified, the absolute value of the number is used.

Examples

The following example creates Random objects with the class constructor that takes a seed parameter and generates a sequence of random integers and doubles. The example illustrates that the same sequence is generated when the Random object is created again with the constructor and seed parameter.

C#

```
// Example of the Random class constructors and Random.NextDouble()
// method.
using System;
using System.Threading;

public class RandomObjectDemo
{
    // Generate random numbers from the specified Random object.
    static void RunIntNDoubleRandoms(Random randObj)
    {
        // Generate the first six random integers.
        for(int j = 0; j < 6; j++)
            Console.Write(" {0,10} ", randObj.Next());
        Console.WriteLine();

        // Generate the first six random doubles.
        for(int j = 0; j < 6; j++)
            Console.Write(" {0:F8} ", randObj.NextDouble());
        Console.WriteLine();
    }

    // Create a Random object with the specified seed.
    static void FixedSeedRandoms(int seed)
    {
        Console.WriteLine(
            "\nRandom numbers from a Random object with " +
            "seed = {0}:", seed);
        Random fixRand = new Random(seed);

        RunIntNDoubleRandoms(fixRand);
    }

    // Create a random object with a timer-generated seed.
    static void AutoSeedRandoms()
    {
        // Wait to allow the timer to advance.
        Thread.Sleep(1);

        Console.WriteLine(
            "\nRandom numbers from a Random object " +
            "with an auto-generated seed:");
        Random autoRand = new Random();
```

```
        RunIntNDoubleRandoms(autoRand);
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of the Random class constructors and " +
            "Random.NextDouble() \n" +
            "generates the following output.\n");
        Console.WriteLine(
            "Create Random objects, and then generate and " +
            "display six integers and \nsix doubles from each.");

        FixedSeedRandoms(123);
        FixedSeedRandoms(123);

        FixedSeedRandoms(456);
        FixedSeedRandoms(456);

        AutoSeedRandoms();
        AutoSeedRandoms();
        AutoSeedRandoms();
    }
}

/*
This example of the Random class constructors and Random.NextDouble()
generates an output similar to the following:

Create Random objects, and then generate and display six integers and
six doubles from each.

Random numbers from a Random object with seed = 123:
 2114319875  1949518561  1596751841  1742987178  1586516133   103755708
 0.01700087  0.14935942  0.19470390  0.63008947  0.90976122  0.49519146

Random numbers from a Random object with seed = 123:
 2114319875  1949518561  1596751841  1742987178  1586516133   103755708
 0.01700087  0.14935942  0.19470390  0.63008947  0.90976122  0.49519146

Random numbers from a Random object with seed = 456:
 2044805024  1323311594  1087799997  1907260840   179380355   120870348
 0.21988117  0.21026556  0.39236514  0.42420498  0.24102703  0.47310170

Random numbers from a Random object with seed = 456:
 2044805024  1323311594  1087799997  1907260840   179380355   120870348
 0.21988117  0.21026556  0.39236514  0.42420498  0.24102703  0.47310170

Random numbers from a Random object with an auto-generated seed:
  380213349   127379247  1969091178  1983029819  1963098450  1648433124
 0.08824121  0.41249688  0.36445811  0.05637512  0.62702451  0.49595560

Random numbers from a Random object with an auto-generated seed:
  861793304  2133528783  1947358439   124230908   921262645  1087892791
 0.56880819  0.42934091  0.60162512  0.74388610  0.99432979  0.30310005
```

```
Random numbers from a Random object with an auto-generated seed:
 1343373259  1992194672  1925625700   412915644  2026910487   527352458
 0.04937517  0.44618494  0.83879212  0.43139707  0.36163507  0.11024451
*/
```

Remarks

Providing an identical seed value to different Random objects causes each instance to produce identical sequences of random numbers. This is often done when testing apps that rely on random number generators.

If your application requires different random number sequences, invoke this constructor repeatedly with different seed values. One way to produce a unique seed value is to make it time-dependent. For example, derive the seed value from the system clock, as the Random() overload does. However, the system clock might not have sufficient resolution to provide different invocations of this constructor with a different seed value. On the .NET Framework, this results in random number generators that generate identical sequences of pseudo-random numbers, as illustrated by the first two Random objects in the following example. To prevent this, apply an algorithm to differentiate the seed value in each invocation, or call the Thread.Sleep method to ensure that you provide each constructor with a different seed value.

C#

```
using System;
using System.Threading;

public class Example
{
   public static void Main()
   {
      Random rand1 = new Random((int) DateTime.Now.Ticks & 0x0000FFFF);
      Random rand2 = new Random((int) DateTime.Now.Ticks & 0x0000FFFF);
      Thread.Sleep(20);
      Random rand3 = new Random((int) DateTime.Now.Ticks & 0x0000FFFF);
      ShowRandomNumbers(rand1);
      ShowRandomNumbers(rand2);
      ShowRandomNumbers(rand3);
   }

   private static void ShowRandomNumbers(Random rand)
   {
      Console.WriteLine();
      byte[] values = new byte[4];
      rand.NextBytes(values);
      foreach (var value in values)
```

```
            Console.Write("{0, 5}", value);

        Console.WriteLine();
    }
}
// The example displays output similar to the following:
//   145  214  177  134  173
//
//   145  214  177  134  173
//
//   126  185  175  249  157
```

Another option is to instantiate a single Random object that you use to generate all the random numbers in your application. This yields slightly better performance, since instantiating a random number generator is fairly expensive.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.Shared Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a thread-safe Random instance that may be used concurrently from any thread.

C#

```
public static Random Shared { get; }
```

Property Value

Random

A Random instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.GetItems Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetItems<T>(ReadOnlySpan<T>, Int32)	Creates an array populated with items chosen at random from the provided set of choices.
GetItems<T>(ReadOnlySpan<T>, Span<T>)	Fills the elements of a specified span with items chosen at random from the provided set of choices.
GetItems<T>(T[], Int32)	Creates an array populated with items chosen at random from the provided set of choices.

GetItems<T>(ReadOnlySpan<T>, Int32)

Creates an array populated with items chosen at random from the provided set of choices.

C#

```
public T[] GetItems<T> (ReadOnlySpan<T> choices, int length);
```

Type Parameters

T

The type of array.

Parameters

choices ReadOnlySpan<T>

The items to use to populate the array.

`length` Int32

The length of array to return.

Returns

T[]

An array populated with random items.

Exceptions

ArgumentException

`choices` is empty.

ArgumentOutOfRangeException

`length` is not zero or a positive number.

Remarks

The method uses Next(Int32) to select items randomly from `choices` by index. This is used to populate a newly-created array.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

GetItems<T>(ReadOnlySpan<T>, Span<T>)

Fills the elements of a specified span with items chosen at random from the provided set of choices.

```
C#

public void GetItems<T> (ReadOnlySpan<T> choices, Span<T> destination);
```

Type Parameters

`T`

The type of span.

Parameters

`choices` ReadOnlySpan<T>

The items to use to populate the span.

`destination` Span<T>

The span to be filled with items.

Exceptions

ArgumentException

`choices` is empty.

Remarks

The method uses Next(Int32) to select items randomly from `choices` by index and populate `destination`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

GetItems<T>(T[], Int32)

Creates an array populated with items chosen at random from the provided set of choices.

```
C#
public T[] GetItems<T> (T[] choices, int length);
```

Type Parameters

`T`

The type of array.

Parameters

`choices` T[]

The items to use to populate the array.

`length` Int32

The length of array to return.

Returns

T[]

An array populated with random items.

Exceptions

ArgumentException

`choices` is empty.

ArgumentNullException

`choices` is `null`.

ArgumentOutOfRangeException

`length` is not zero or a positive number.

Remarks

The method uses Next(Int32) to select items randomly from `choices` by index. This is used to populate a newly-created array.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Random.Next Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a random integer.

Overloads

Expand table

Next()	Returns a non-negative random integer.
Next(Int32)	Returns a non-negative random integer that is less than the specified maximum.
Next(Int32, Int32)	Returns a random integer that is within a specified range.

Next()

Returns a non-negative random integer.

C#

```
public virtual int Next ();
```

Returns

Int32

A 32-bit signed integer that is greater than or equal to 0 and less than Int32.MaxValue.

Examples

The following example makes repeated calls to the Next method to generate a specific number of random numbers requested by the user. The Console.ReadLine method is used to get customer input.

C#

```
Random rnd = new Random();

Console.WriteLine("Generating 10 random numbers:");

for (uint ctr = 1; ctr <= 10; ctr++)
   Console.WriteLine($"{rnd.Next(),15:N0}");

// The example displays output like the following:
//
//     Generating 10 random numbers:
//         1,733,189,596
//           566,518,090
//         1,166,108,546
//         1,931,426,514
//         1,532,939,448
//           762,207,767
//           815,074,920
//         1,521,208,785
//         1,950,436,671
//         1,266,596,666
```

The following example derives a class from Random to generate a sequence of random numbers whose distribution differs from the uniform distribution generated by the Sample method of the base class. It overrides the Sample method to provide the distribution of random numbers, and overrides the Random.Next method to use series of random numbers.

C#

```
using System;

// This derived class converts the uniformly distributed random
// numbers generated by base.Sample() to another distribution.
public class RandomProportional : Random
{
    // The Sample method generates a distribution proportional to the
value
    // of the random numbers, in the range [0.0, 1.0].
    protected override double Sample()
    {
        return Math.Sqrt(base.Sample());
    }

    public override int Next()
```

```
    {
        return (int) (Sample() * int.MaxValue);
    }
}

public class RandomSampleDemo
{
    static void Main()
    {
        const int rows = 4, cols = 6;
        const int runCount = 1000000;
        const int distGroupCount = 10;
        const double intGroupSize =
            ((double)int.MaxValue + 1.0) / (double)distGroupCount;

        RandomProportional randObj = new RandomProportional();

        int[ ]      intCounts = new int[ distGroupCount ];
        int[ ]      realCounts = new int[ distGroupCount ];

        Console.WriteLine(
            "\nThe derived RandomProportional class overrides " +
            "the Sample method to \ngenerate random numbers " +
            "in the range [0.0, 1.0]. The distribution \nof " +
            "the numbers is proportional to their numeric values. " +
            "For example, \nnumbers are generated in the " +
            "vicinity of 0.75 with three times the \n" +
            "probability of those generated near 0.25.");
        Console.WriteLine(
            "\nRandom doubles generated with the NextDouble() " +
            "method:\n");

        // Generate and display [rows * cols] random doubles.
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
                Console.Write("{0,12:F8}", randObj.NextDouble());
            Console.WriteLine();
        }

        Console.WriteLine(
            "\nRandom integers generated with the Next() " +
            "method:\n");

        // Generate and display [rows * cols] random integers.
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
                Console.Write("{0,12}", randObj.Next());
            Console.WriteLine();
        }

        Console.WriteLine(
            "\nTo demonstrate the proportional distribution, " +
            "{0:N0} random \nintegers and doubles are grouped " +
```

```
            "into {1} equal value ranges. This \n" +
            "is the count of values in each range:\n",
            runCount, distGroupCount);
        Console.WriteLine(
            "{0,21}{1,10}{2,20}{3,10}", "Integer Range",
            "Count", "Double Range", "Count");
        Console.WriteLine(
            "{0,21}{1,10}{2,20}{3,10}", "-------------",
            "-----", "------------", "-----");

        // Generate random integers and doubles, and then count
        // them by group.
        for (int i = 0; i < runCount; i++)
        {
            intCounts[ (int)((double)randObj.Next() /
                intGroupSize) ]++;
            realCounts[ (int)(randObj.NextDouble() *
                (double)distGroupCount) ]++;
        }

        // Display the count of each group.
        for (int i = 0; i < distGroupCount; i++)
            Console.WriteLine(
                "{0,10}-{1,10}{2,10:N0}{3,12:N5}-{4,7:N5}{5,10:N0}",
                (int)((double)i * intGroupSize),
                (int)((double)(i + 1) * intGroupSize - 1.0),
                intCounts[ i ],
                ((double)i) / (double)distGroupCount,
                ((double)(i + 1)) / (double)distGroupCount,
                realCounts[ i ]);
    }
}

/*
This example of Random.Sample() displays output similar to the following:

   The derived RandomProportional class overrides the Sample method to
   generate random numbers in the range [0.0, 1.0). The distribution
   of the numbers is proportional to the number values. For example,
   numbers are generated in the vicinity of 0.75 with three times the
   probability of those generated near 0.25.

   Random doubles generated with the NextDouble() method:

     0.59455719  0.17589882  0.83134398  0.35795862  0.91467727
0.54022658
     0.93716947  0.54817519  0.94685080  0.93705478  0.18582318
0.71272428
     0.77708682  0.95386216  0.70412393  0.86099417  0.08275804
0.79108316
     0.71019941  0.84205103  0.41685082  0.58186880  0.89492302
0.73067715

   Random integers generated with the Next() method:
```

```
     1570755704  1279192549  1747627711  1705700211  1372759203
1849655615
     2046235980  1210843924  1554274149  1307936697  1480207570
1057595022
      337854215   844109928  2028310798  1386669369  2073517658
1291729809
     1537248240  1454198019  1934863511  1640004334  2032620207
534654791

   To demonstrate the proportional distribution, 1,000,000 random
   integers and doubles are grouped into 10 equal value ranges. This
   is the count of values in each range:

           Integer Range     Count        Double Range     Count
           -------------     -----        ------------     -----
            0- 214748363    10,079     0.00000-0.10000    10,148
    214748364- 429496728    29,835     0.10000-0.20000    29,849
    429496729- 644245093    49,753     0.20000-0.30000    49,948
    644245094- 858993458    70,325     0.30000-0.40000    69,656
    858993459-1073741823    89,906     0.40000-0.50000    90,337
   1073741824-1288490187   109,868     0.50000-0.60000   110,225
   1288490188-1503238552   130,388     0.60000-0.70000   129,986
   1503238553-1717986917   149,231     0.70000-0.80000   150,428
   1717986918-1932735282   170,234     0.80000-0.90000   169,610
   1932735283-2147483647   190,381     0.90000-1.00000   189,813
*/
```

Remarks

Random.Next generates a random number whose value ranges from 0 to less than Int32.MaxValue. To generate a random number whose value ranges from 0 to some other positive number, use the Random.Next(Int32) method overload. To generate a random number within a different range, use the Random.Next(Int32, Int32) method overload.

Notes to Inheritors

Starting with the .NET Framework version 2.0, if you derive a class from Random and override the Sample() method, the distribution provided by the derived class implementation of the Sample() method is not used in calls to the base class implementation of the Next() method. Instead, the uniform distribution returned by the base Random class is used. This behavior improves the overall performance of the Random class. To modify this behavior to call the Sample() method in the derived class, you must also override the Next() method.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Next(Int32)

Returns a non-negative random integer that is less than the specified maximum.

C#

```
public virtual int Next (int maxValue);
```

Parameters

`maxValue` Int32

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

Int32

A 32-bit signed integer that is greater than or equal to 0, and less than `maxValue`; that is, the range of return values ordinarily includes 0 but not `maxValue`. However, if `maxValue` equals 0, 0 is returned.

Exceptions

ArgumentOutOfRangeException

`maxValue` is less than 0.

Examples

The following example generates random integers with various overloads of the Next method.

C#

```
Console.WriteLine(
    "This example of the Random.Next() methods\n" +
    "generates the following output.\n");
Console.WriteLine(
    "Create Random objects all with the same seed and " +
    "generate\nsequences of numbers with different " +
    "bounds. Note the effect\nthat the various " +
    "combinations of bounds have on the sequences.");

NoBoundsRandoms(234);

UpperBoundRandoms(234, Int32.MaxValue);
UpperBoundRandoms(234, 2000000000);
UpperBoundRandoms(234, 200000000);

BothBoundsRandoms(234, 0, Int32.MaxValue);
BothBoundsRandoms(234, Int32.MinValue, Int32.MaxValue);
BothBoundsRandoms(234, -2000000000, 2000000000);
BothBoundsRandoms(234, -200000000, 200000000);
BothBoundsRandoms(234, -2000, 2000);

// Generate random numbers with no bounds specified.
void NoBoundsRandoms(int seed)
{
    Console.WriteLine(
        "\nRandom object, seed = {0}, no bounds:", seed);
    Random randObj = new Random(seed);

    // Generate six random integers from 0 to int.MaxValue.
    for (int j = 0; j < 6; j++)
        Console.Write("{0,11} ", randObj.Next());
    Console.WriteLine();
}

// Generate random numbers with an upper bound specified.
void UpperBoundRandoms(int seed, int upper)
{
    Console.WriteLine(
        "\nRandom object, seed = {0}, upper bound = {1}:",
        seed, upper);
    Random randObj = new Random(seed);

    // Generate six random integers from 0 to the upper bound.
```

```
    for (int j = 0; j < 6; j++)
        Console.Write("{0,11} ", randObj.Next(upper));
    Console.WriteLine();
}

// Generate random numbers with both bounds specified.
void BothBoundsRandoms(int seed, int lower, int upper)
{
    Console.WriteLine(
        "\nRandom object, seed = {0}, lower = {1}, " +
        "upper = {2}:", seed, lower, upper);
    Random randObj = new Random(seed);

    // Generate six random integers from the lower to
    // upper bounds.
    for (int j = 0; j < 6; j++)
        Console.Write("{0,11} ",
            randObj.Next(lower, upper));
    Console.WriteLine();
}

/*
This example of the Random.Next() methods
generates the following output.

Create Random objects all with the same seed and generate
sequences of numbers with different bounds. Note the effect
that the various combinations of bounds have on the sequences.

Random object, seed = 234, no bounds:
2091148258  1024955023   711273344  1081917183  1833298756   109460588

Random object, seed = 234, upper bound = 2147483647:
2091148258  1024955023   711273344  1081917183  1833298756   109460588

Random object, seed = 234, upper bound = 2000000000:
1947533580   954563751   662424922  1007613896  1707392518   101943116

Random object, seed = 234, upper bound = 200000000:
194753358    95456375    66242492   100761389   170739251    10194311

Random object, seed = 234, lower = 0, upper = 2147483647:
2091148258  1024955023   711273344  1081917183  1833298756   109460588

Random object, seed = 234, lower = -2147483648, upper = 2147483647:
2034812868   -97573602  -724936960    16350718  1519113864 -1928562472

Random object, seed = 234, lower = -2000000000, upper = 2000000000:
1895067160   -90872498  -675150156    15227793  1414785036 -1796113767

Random object, seed = 234, lower = -200000000, upper = 200000000:
189506716    -9087250   -67515016     1522779   141478503  -179611377

Random object, seed = 234, lower = -2000, upper = 2000:
```

```
    1895          -91         -676           15         1414        -1797
*/
```

The following example generates a random integer that it uses as an index to retrieve a string value from an array. Because the highest index of the array is one less than its length, the value of the Array.Length property is supplied as a the `maxValue` parameter.

C#

```
Random rnd = new Random();
string[] malePetNames = { "Rufus", "Bear", "Dakota", "Fido",
                          "Vanya", "Samuel", "Koani", "Volodya",
                          "Prince", "Yiska" };
string[] femalePetNames = { "Maggie", "Penny", "Saya", "Princess",
                            "Abby", "Laila", "Sadie", "Olivia",
                            "Starlight", "Talla" };

// Generate random indexes for pet names.
int mIndex = rnd.Next(malePetNames.Length);
int fIndex = rnd.Next(femalePetNames.Length);

// Display the result.
Console.WriteLine("Suggested pet name of the day: ");
Console.WriteLine("   For a male:     {0}", malePetNames[mIndex]);
Console.WriteLine("   For a female:   {0}", femalePetNames[fIndex]);

// The example displays output similar to the following:
//       Suggested pet name of the day:
//          For a male:     Koani
//          For a female:   Maggie
```

Remarks

The Next(Int32) overload returns random integers that range from 0 to `maxValue` - 1. However, if `maxValue` is 0, the method returns 0.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Next(Int32, Int32)

Returns a random integer that is within a specified range.

C#

```
public virtual int Next (int minValue, int maxValue);
```

Parameters

`minValue` Int32

The inclusive lower bound of the random number returned.

`maxValue` Int32

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

Int32

A 32-bit signed integer greater than or equal to `minValue` and less than `maxValue`; that is, the range of return values includes `minValue` but not `maxValue`. If `minValue` equals `maxValue`, `minValue` is returned.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

Examples

The following example uses the Random.Next(Int32, Int32) method to generate random integers with three distinct ranges. Note that the exact output from the example depends on the system-supplied seed value passed to the Random class constructor.

C#

```
Random rnd = new Random();

Console.WriteLine("\n20 random integers from -100 to 100:");
for (int ctr = 1; ctr <= 20; ctr++)
{
   Console.Write("{0,6}", rnd.Next(-100, 101));
   if (ctr % 5 == 0) Console.WriteLine();
}

Console.WriteLine("\n20 random integers from 1000 to 10000:");
for (int ctr = 1; ctr <= 20; ctr++)
{
   Console.Write("{0,8}", rnd.Next(1000, 10001));
   if (ctr % 5 == 0) Console.WriteLine();
}

Console.WriteLine("\n20 random integers from 1 to 10:");
for (int ctr = 1; ctr <= 20; ctr++)
{
   Console.Write("{0,6}", rnd.Next(1, 11));
   if (ctr % 5 == 0) Console.WriteLine();
}

// The example displays output similar to the following:
//       20 random integers from -100 to 100:
//           65   -95   -10    90   -35
//          -83   -16   -15   -19    41
//          -67   -93    40    12    62
//          -80   -95    67   -81   -21
//
//       20 random integers from 1000 to 10000:
//           4857    9897    4405    6606    1277
//           9238    9113    5151    8710    1187
//           2728    9746    1719    3837    3736
//           8191    6819    4923    2416    3028
//
//       20 random integers from 1 to 10:
//            9     8     5     9     9
//            9     1     2     3     8
//            1     4     8    10     5
//            9     7     9    10     5
```

The following example generates a random integer that it uses as an index to retrieve a string value from an array. Because the highest index of the array is one less than its length, the value of the Array.Length property is supplied as a the `maxValue` parameter.

C#

```
Random rnd = new Random();
string[] malePetNames = { "Rufus", "Bear", "Dakota", "Fido",
                          "Vanya", "Samuel", "Koani", "Volodya",
                          "Prince", "Yiska" };
string[] femalePetNames = { "Maggie", "Penny", "Saya", "Princess",
                            "Abby", "Laila", "Sadie", "Olivia",
                            "Starlight", "Talla" };

// Generate random indexes for pet names.
int mIndex = rnd.Next(0, malePetNames.Length);
int fIndex = rnd.Next(0, femalePetNames.Length);

// Display the result.
Console.WriteLine("Suggested pet name of the day: ");
Console.WriteLine("   For a male:     {0}", malePetNames[mIndex]);
Console.WriteLine("   For a female:   {0}", femalePetNames[fIndex]);

// The example displays the following output:
//       Suggested pet name of the day:
//          For a male:     Koani
//          For a female:   Maggie
```

Remarks

The Next(Int32, Int32) overload returns random integers that range from `minValue` to `maxValue` - 1. However, if `maxValue` equals `minValue`, the method returns `minValue`.

Unlike the other overloads of the Next method, which return only non-negative values, this method can return a negative random integer.

Notes to Inheritors

Starting with the .NET Framework version 2.0, if you derive a class from Random and override the Sample() method, the distribution provided by the derived class implementation of the Sample() method is not used in calls to the base class implementation of the Next(Int32, Int32) method overload if the difference between the `minValue` and `maxValue` parameters is greater than Int32.MaxValue. Instead, the uniform distribution returned by the base Random class is used. This behavior improves the overall performance of the Random class. To modify this behavior to

call the Sample() method in the derived class, you must also override the Next(Int32, Int32) method overload.

See also

- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.NextBytes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

NextBytes(Span<Byte>)	Fills the elements of a specified span of bytes with random numbers.
NextBytes(Byte[])	Fills the elements of a specified array of bytes with random numbers.

NextBytes(Span<Byte>)

Fills the elements of a specified span of bytes with random numbers.

C#

```
public virtual void NextBytes (Span<byte> buffer);
```

Parameters

`buffer` Span<Byte>

The array to be filled with random numbers.

Remarks

Each element of the span of bytes is set to a random number greater than or equal to 0 and less than or equal to MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

NextBytes(Byte[])

Fills the elements of a specified array of bytes with random numbers.

C#

```
public virtual void NextBytes (byte[] buffer);
```

Parameters

`buffer` Byte[]

The array to be filled with random numbers.

Exceptions

ArgumentNullException

`buffer` is `null`.

Examples

The following example demonstrates how to use the NextBytes method to fill an array of bytes with random byte values.

C#

```
Random rnd = new Random();
Byte[] b = new Byte[10];
rnd.NextBytes(b);
Console.WriteLine("The Random bytes are: ");
for (int i = 0; i <= b.GetUpperBound(0); i++)
    Console.WriteLine("{0}: {1}", i, b[i]);

// The example displays output similar to the following:
//       The Random bytes are:
//       0: 131
//       1: 96
//       2: 226
//       3: 213
//       4: 176
```

```
//       5: 208
//       6: 99
//       7: 89
//       8: 226
//       9: 194
```

Remarks

Each element of the array of bytes is set to a random number greater than or equal to 0, and less than or equal to MaxValue.

For example, to generate a cryptographically secured random number suitable for creating a random password, use a method such as RNGCryptoServiceProvider.GetBytes.

Notes to Inheritors

Starting with the .NET Framework version 2.0, if you derive a class from Random and override the Sample() method, the distribution provided by the derived class implementation of the Sample() method is not used in calls to the base class implementation of the NextBytes(Byte[]) method. Instead, the uniform distribution returned by the base Random class is used. This behavior improves the overall performance of the Random class. To modify this behavior to call the Sample() method in the derived class, you must also override the NextBytes(Byte[]) method.

See also

- Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Random.NextDouble Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0.

C#

```
public virtual double NextDouble ();
```

Returns

Double

A double-precision floating point number that is greater than or equal to 0.0, and less than 1.0.

Examples

The following example uses the NextDouble method to generate sequences of random doubles.

C#

```
// Example of the Random class constructors and Random.NextDouble()
// method.
using System;
using System.Threading;

public class RandomObjectDemo
{
    // Generate random numbers from the specified Random object.
    static void RunIntNDoubleRandoms(Random randObj)
    {
        // Generate the first six random integers.
        for(int j = 0; j < 6; j++)
            Console.Write(" {0,10} ", randObj.Next());
        Console.WriteLine();
```

```
        // Generate the first six random doubles.
        for(int j = 0; j < 6; j++)
            Console.Write(" {0:F8} ", randObj.NextDouble());
        Console.WriteLine();
    }

    // Create a Random object with the specified seed.
    static void FixedSeedRandoms(int seed)
    {
        Console.WriteLine(
            "\nRandom numbers from a Random object with " +
            "seed = {0}:", seed);
        Random fixRand = new Random(seed);

        RunIntNDoubleRandoms(fixRand);
    }

    // Create a random object with a timer-generated seed.
    static void AutoSeedRandoms()
    {
        // Wait to allow the timer to advance.
        Thread.Sleep(1);

        Console.WriteLine(
            "\nRandom numbers from a Random object " +
            "with an auto-generated seed:");
        Random autoRand = new Random();

        RunIntNDoubleRandoms(autoRand);
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of the Random class constructors and " +
            "Random.NextDouble() \n" +
            "generates the following output.\n");
        Console.WriteLine(
            "Create Random objects, and then generate and " +
            "display six integers and \nsix doubles from each.");

        FixedSeedRandoms(123);
        FixedSeedRandoms(123);

        FixedSeedRandoms(456);
        FixedSeedRandoms(456);

        AutoSeedRandoms();
        AutoSeedRandoms();
        AutoSeedRandoms();
    }
}

/*
This example of the Random class constructors and Random.NextDouble()
```

```
generates an output similar to the following:

Create Random objects, and then generate and display six integers and
six doubles from each.

Random numbers from a Random object with seed = 123:
 2114319875  1949518561  1596751841  1742987178  1586516133   103755708
 0.01700087  0.14935942  0.19470390  0.63008947  0.90976122  0.49519146

Random numbers from a Random object with seed = 123:
 2114319875  1949518561  1596751841  1742987178  1586516133   103755708
 0.01700087  0.14935942  0.19470390  0.63008947  0.90976122  0.49519146

Random numbers from a Random object with seed = 456:
 2044805024  1323311594  1087799997  1907260840   179380355   120870348
 0.21988117  0.21026556  0.39236514  0.42420498  0.24102703  0.47310170

Random numbers from a Random object with seed = 456:
 2044805024  1323311594  1087799997  1907260840   179380355   120870348
 0.21988117  0.21026556  0.39236514  0.42420498  0.24102703  0.47310170

Random numbers from a Random object with an auto-generated seed:
  380213349   127379247  1969091178  1983029819  1963098450  1648433124
 0.08824121  0.41249688  0.36445811  0.05637512  0.62702451  0.49595560

Random numbers from a Random object with an auto-generated seed:
  861793304  2133528783  1947358439   124230908   921262645  1087892791
 0.56880819  0.42934091  0.60162512  0.74388610  0.99432979  0.30310005

Random numbers from a Random object with an auto-generated seed:
 1343373259  1992194672  1925625700   412915644  2026910487   527352458
 0.04937517  0.44618494  0.83879212  0.43139707  0.36163507  0.11024451
*/
```

The following example calls the NextDouble method to generate 100 random numbers and displays their frequency distribution.

C#

```
int[] frequency = new int[10];
double number;
Random rnd = new Random();

for (int ctr = 0; ctr <= 99; ctr++) {
   number = rnd.NextDouble();
   frequency[(int) Math.Floor(number*10)] ++;
}
Console.WriteLine("Distribution of Random Numbers:");
for (int ctr = frequency.GetLowerBound(0); ctr <=
frequency.GetUpperBound(0); ctr++)
   Console.WriteLine("0.{0}0-0.{0}9       {1}", ctr, frequency[ctr]);

// The following example displays output similar to the following:
```

```
//       Distribution of Random Numbers:
//       0.00-0.09       16
//       0.10-0.19       8
//       0.20-0.29       8
//       0.30-0.39       11
//       0.40-0.49       9
//       0.50-0.59       6
//       0.60-0.69       13
//       0.70-0.79       6
//       0.80-0.89       9
//       0.90-0.99       14
```

Remarks

The actual upper bound of the random number returned by this method is 0.99999999999999978.

To retrieve random floating-point values within a range other than 0.0 and 1.0, see the "Retrieve floating-point values in a specified range" section of the Random class topic.

This method is the public version of the protected method, Sample.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.NextInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

NextInt64()	Returns a non-negative random integer.
NextInt64(Int64)	Returns a non-negative random integer that is less than the specified maximum.
NextInt64(Int64, Int64)	Returns a random integer that is within a specified range.

NextInt64()

Returns a non-negative random integer.

C#

```
public virtual long NextInt64 ();
```

Returns

Int64

A 64-bit signed integer that is greater than or equal to 0 and less than Int64.MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

NextInt64(Int64)

Returns a non-negative random integer that is less than the specified maximum.

```
C#
public virtual long NextInt64 (long maxValue);
```

Parameters

`maxValue` Int64

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

Int64

A 64-bit signed integer that is greater than or equal to 0, and less than `maxValue`; that is, the range of return values ordinarily includes 0 but not `maxValue`. However, if `maxValue` equals 0, `maxValue` is returned.

Exceptions

ArgumentOutOfRangeException

`maxValue` is less than 0.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

NextInt64(Int64, Int64)

Returns a random integer that is within a specified range.

```
C#
public virtual long NextInt64 (long minValue, long maxValue);
```

Parameters

`minValue` Int64

The inclusive lower bound of the random number returned.

`maxValue` Int64

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

Int64

A 64-bit signed integer greater than or equal to `minValue` and less than `maxValue`; that is, the range of return values includes `minValue` but not `maxValue`. If minValue equals `maxValue`, `minValue` is returned.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Random.NextSingle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0.

C#

```
public virtual float NextSingle ();
```

Returns

Single

A single-precision floating point number that is greater than or equal to 0.0, and less than 1.0.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.Sample Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a random floating-point number between 0.0 and 1.0.

C#

```
protected virtual double Sample ();
```

Returns

Double

A double-precision floating point number that is greater than or equal to 0.0, and less than 1.0.

Examples

The following example derives a class from Random and overrides the Sample method to generate a distribution of random numbers. This distribution is different than the uniform distribution generated by the Sample method of the base class.

C#

```
using System;

// This derived class converts the uniformly distributed random
// numbers generated by base.Sample() to another distribution.
public class RandomProportional : Random
{
    // The Sample method generates a distribution proportional to the value
    // of the random numbers, in the range [0.0, 1.0].
    protected override double Sample()
    {
        return Math.Sqrt(base.Sample());
    }

    public override int Next()
    {
```

```
        return (int) (Sample() * int.MaxValue);
    }
}

public class RandomSampleDemo
{
    static void Main()
    {
        const int rows = 4, cols = 6;
        const int runCount = 1000000;
        const int distGroupCount = 10;
        const double intGroupSize =
            ((double)int.MaxValue + 1.0) / (double)distGroupCount;

        RandomProportional randObj = new RandomProportional();

        int[ ]      intCounts = new int[ distGroupCount ];
        int[ ]      realCounts = new int[ distGroupCount ];

        Console.WriteLine(
            "\nThe derived RandomProportional class overrides " +
            "the Sample method to \ngenerate random numbers " +
            "in the range [0.0, 1.0]. The distribution \nof " +
            "the numbers is proportional to their numeric values. " +
            "For example, \nnumbers are generated in the " +
            "vicinity of 0.75 with three times the \n" +
            "probability of those generated near 0.25.");
        Console.WriteLine(
            "\nRandom doubles generated with the NextDouble() " +
            "method:\n");

        // Generate and display [rows * cols] random doubles.
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
                Console.Write("{0,12:F8}", randObj.NextDouble());
            Console.WriteLine();
        }

        Console.WriteLine(
            "\nRandom integers generated with the Next() " +
            "method:\n");

        // Generate and display [rows * cols] random integers.
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
                Console.Write("{0,12}", randObj.Next());
            Console.WriteLine();
        }

        Console.WriteLine(
            "\nTo demonstrate the proportional distribution, " +
            "{0:N0} random \nintegers and doubles are grouped " +
            "into {1} equal value ranges. This \n" +
```

```
            "is the count of values in each range:\n",
            runCount, distGroupCount);
        Console.WriteLine(
            "{0,21}{1,10}{2,20}{3,10}", "Integer Range",
            "Count", "Double Range", "Count");
        Console.WriteLine(
            "{0,21}{1,10}{2,20}{3,10}", "-------------",
            "-----", "------------", "-----");

        // Generate random integers and doubles, and then count
        // them by group.
        for (int i = 0; i < runCount; i++)
        {
            intCounts[ (int)((double)randObj.Next() /
                intGroupSize) ]++;
            realCounts[ (int)(randObj.NextDouble() *
                (double)distGroupCount) ]++;
        }

        // Display the count of each group.
        for (int i = 0; i < distGroupCount; i++)
            Console.WriteLine(
                "{0,10}-{1,10}{2,10:N0}{3,12:N5}-{4,7:N5}{5,10:N0}",
                (int)((double)i * intGroupSize),
                (int)((double)(i + 1) * intGroupSize - 1.0),
                intCounts[ i ],
                ((double)i) / (double)distGroupCount,
                ((double)(i + 1)) / (double)distGroupCount,
                realCounts[ i ]);
    }
}

/*
This example of Random.Sample() displays output similar to the following:

   The derived RandomProportional class overrides the Sample method to
   generate random numbers in the range [0.0, 1.0). The distribution
   of the numbers is proportional to the number values. For example,
   numbers are generated in the vicinity of 0.75 with three times the
   probability of those generated near 0.25.

   Random doubles generated with the NextDouble() method:

     0.59455719  0.17589882  0.83134398  0.35795862  0.91467727  0.54022658
     0.93716947  0.54817519  0.94685080  0.93705478  0.18582318  0.71272428
     0.77708682  0.95386216  0.70412393  0.86099417  0.08275804  0.79108316
     0.71019941  0.84205103  0.41685082  0.58186880  0.89492302  0.73067715

   Random integers generated with the Next() method:

     1570755704  1279192549  1747627711  1705700211  1372759203  1849655615
     2046235980  1210843924  1554274149  1307936697  1480207570  1057595022
      337854215   844109928  2028310798  1386669369  2073517658  1291729809
     1537248240  1454198019  1934863511  1640004334  2032620207   534654791
```

```
    To demonstrate the proportional distribution, 1,000,000 random
    integers and doubles are grouped into 10 equal value ranges. This
    is the count of values in each range:

            Integer Range     Count        Double Range     Count
            -------------     -----        ------------     -----
             0- 214748363    10,079     0.00000-0.10000    10,148
     214748364- 429496728    29,835     0.10000-0.20000    29,849
     429496729- 644245093    49,753     0.20000-0.30000    49,948
     644245094- 858993458    70,325     0.30000-0.40000    69,656
     858993459-1073741823    89,906     0.40000-0.50000    90,337
    1073741824-1288490187   109,868     0.50000-0.60000   110,225
    1288490188-1503238552   130,388     0.60000-0.70000   129,986
    1503238553-1717986917   149,231     0.70000-0.80000   150,428
    1717986918-1932735282   170,234     0.80000-0.90000   169,610
    1932735283-2147483647   190,381     0.90000-1.00000   189,813
*/
```

Remarks

To produce a different random distribution or a different random number generator principle, derive a class from the Random class and override the Sample method.

> ⓘ **Important**
>
> The **Sample** method is `protected`, which means that it is accessible only within the **Random** class and its derived classes. To generate a random number between 0 and 1 from a **Random** instance, call the **NextDouble** method.

Notes to Inheritors

Starting with the .NET Framework version 2.0, if you derive a class from Random and override the Sample() method, the distribution provided by the derived class implementation of the Sample() method is not used in calls to the base class implementation of the following methods:

- The NextBytes(Byte[]) method.
- The Next() method.
- The Next(Int32, Int32) method, if (`maxValue` - `minValue`) is greater than Int32.MaxValue.

Instead, the uniform distribution provided by the base Random class is used. This behavior improves the overall performance of the Random class. To modify this behavior to call the implementation of the Sample() method in the derived class, you must also override the behavior of these three members. The example provides an illustration.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- NextDouble()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Random.Shuffle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Shuffle<T>(Span<T>)	Performs an in-place shuffle of a span.
Shuffle<T>(T[])	Performs an in-place shuffle of an array.

Shuffle<T>(Span<T>)

Performs an in-place shuffle of a span.

```
C#
public void Shuffle<T> (Span<T> values);
```

Type Parameters

T

The type of span.

Parameters

values Span<T>

The span to shuffle.

Remarks

This method uses Next(Int32, Int32) to choose values for shuffling. This method is an O(n) operation.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Shuffle<T>(T[])

Performs an in-place shuffle of an array.

```
C#
public void Shuffle<T> (T[] values);
```

Type Parameters

`T`

The type of array.

Parameters

`values` T[]

The array to shuffle.

Exceptions

ArgumentNullException

`values` is `null`.

Remarks

This method uses Next(Int32, Int32) to choose values for shuffling. This method is an O(n) operation.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Range Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a range that has start and end indexes.

C#

```
public readonly struct Range : IEquatable<Range>
```

Inheritance Object → ValueType → Range

Implements IEquatable<Range>

Remarks

`Range` is used by the C# compiler to support the range syntax:

C#

```
int[] someArray = new int[5] { 1, 2, 3, 4, 5 };
int[] subArray1 = someArray[0..2];               // { 1, 2 }
int[] subArray2 = someArray[1..^0];              // { 2, 3, 4, 5 }
```

Constructors

Expand table

Range(Index, Index)	Instantiates a new Range instance with the specified starting and ending indexes.

Properties

Expand table

All	Gets a Range object that starts from the first element to the end.
End	Gets an Index that represents the exclusive end index of the range.
Start	Gets the inclusive start index of the Range.

Methods

Expand table

EndAt(Index)	Creates a Range object starting from the first element in the collection to a specified end index.
Equals(Object)	Returns a value that indicates whether the current instance is equal to a specified object.
Equals(Range)	Returns a value that indicates whether the current instance is equal to another Range object.
GetHashCode()	Returns the hash code for this instance.
GetOffsetAndLength(Int32)	Calculates the start offset and length of the range object using a collection length.
StartAt(Index)	Returns a new Range instance starting from a specified start index to the end of the collection.
ToString()	Returns the string representation of the current Range object.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Range(Index, Index) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Instantiates a new Range instance with the specified starting and ending indexes.

C#

```
public Range (Index start, Index end);
```

Parameters

`start` Index

The inclusive start index of the range.

`end` Index

The exclusive end index of the range.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Range.All Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a Range object that starts from the first element to the end.

C#

```
public static Range All { get; }
```

Property Value

Range

A range from the start to the end.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Range.End Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an Index that represents the exclusive end index of the range.

C#

```
public Index End { get; }
```

Property Value

Index

The end index of the range.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Range.Start Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the inclusive start index of the Range.

C#

```
public Index Start { get; }
```

Property Value

Index

The inclusive start index of the range.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Range.EndAt(Index) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a Range object starting from the first element in the collection to a specified end index.

```
C#
public static Range EndAt (Index end);
```

Parameters

end Index

The position of the last element up to which the Range object will be created.

Returns

Range

A range that starts from the first element to `end`.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Range.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current instance is equal to a specified object.
Equals(Range)	Returns a value that indicates whether the current instance is equal to another Range object.

Equals(Object)

Returns a value that indicates whether the current instance is equal to a specified object.

```
C#
public override bool Equals (object? value);
```

Parameters

`value` Object

An object to compare with this Range object.

Returns

Boolean

`true` if `value` is of type Range and is equal to the current instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Equals(Range)

Returns a value that indicates whether the current instance is equal to another Range object.

```
C#
public bool Equals (Range other);
```

Parameters

`other` Range

A Range object to compare with this Range object.

Returns

Boolean

`true` if the current instance is equal to `other`; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Range.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Range.GetOffsetAndLength(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the start offset and length of the range object using a collection length.

```
C#
public (int Offset, int Length) GetOffsetAndLength (int length);
```

Parameters

`length` Int32

A positive integer that represents the length of the collection that the range will be used with.

Returns

ValueTuple<Int32,Int32>

The start offset and length of the range.

Exceptions

ArgumentOutOfRangeException

`length` is outside the bounds of the current range.

Remarks

For performance reasons, this method does't validate `length` to ensure that it is not negative. It does ensure that `length` is within the current `Range` instance.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Range.StartAt(Index) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new Range instance starting from a specified start index to the end of the collection.

```
C#
public static Range StartAt (Index start);
```

Parameters

`start` Index

The position of the first element from which the Range will be created.

Returns

Range

A range from `start` to the end of the collection.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Range.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of the current Range object.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the range.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RankException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an array with the wrong number of dimensions is passed to a method.

C#

```
public class RankException : SystemException
```

Inheritance Object → Exception → SystemException → RankException

Remarks

RankException uses the HRESULT COR_E_RANK, that has the value 0x80131517.

For a list of initial property values for an instance of RankException, see the RankException constructors.

Constructors

Expand table

RankException()	Initializes a new instance of the RankException class.
RankException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the RankException class with serialized data.
RankException(String)	Initializes a new instance of the RankException class with a specified error message.
RankException(String, Exception)	Initializes a new instance of the RankException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

RankException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the RankException class.

Overloads

Expand table

RankException()	Initializes a new instance of the RankException class.
RankException(String)	Initializes a new instance of the RankException class with a specified error message.
RankException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the RankException class with serialized data.
RankException(String, Exception)	Initializes a new instance of the RankException class with a specified error message and a reference to the inner exception that is the cause of this exception.

RankException()

Initializes a new instance of the RankException class.

C#

```
public RankException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The two arrays must have the

same number of dimensions." This message takes into account the current system culture.

The following table shows the initial property values for an instance of RankException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

RankException(String)

Initializes a new instance of the RankException class with a specified error message.

C#

```
public RankException (string? message);
```

Parameters

`message` String

A String that describes the error.

Remarks

The content of the `message` parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of RankException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

RankException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the RankException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected RankException (System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

RankException(String, Exception)

Initializes a new instance of the RankException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public RankException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of RankException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a contiguous region of memory, similar to ReadOnlySpan<T>. Unlike ReadOnlySpan<T>, it is not a byref-like type.

C#

```
public readonly struct ReadOnlyMemory<T> : IEquatable<ReadOnlyMemory<T>>
```

Type Parameters

T

The object type from which the contiguous region of memory will be read.

Inheritance Object → ValueType → ReadOnlyMemory<T>

Implements IEquatable<ReadOnlyMemory<T>>

Constructors

Expand table

ReadOnlyMemory<T>(T[])	Creates a new memory region over the entirety of the target array.
ReadOnlyMemory<T>(T[], Int32, Int32)	Creates a new memory region over the portion of the target array beginning at a specified position and including a specified number of elements.

Properties

Expand table

Empty	Gets an empty ReadOnlyMemory<T>.
IsEmpty	Gets a value that indicates whether this read-only memory region is empty.
Length	Gets the number of items in the memory region.
Span	Gets a span from the memory region.

Methods

Expand table

CopyTo(Memory<T>)	Copies the contents of the read-only memory region into a destination memory region.
Equals(Object)	Determines whether the specified object is equal to the current read-only memory region.
Equals(ReadOnlyMemory<T>)	Determines whether the current instance and a specified ReadOnlyMemory<T> objects are equal.
GetHashCode()	Returns the hash code for this ReadOnlyMemory<T>.
Pin()	Creates a handle for the memory.
Slice(Int32)	Forms a slice out of the given memory region, beginning at a specified position and continuing to its end.
Slice(Int32, Int32)	Forms a slice out of the given memory region starting at `start` position for `length` elements.
ToArray()	Copies the contents from the memory region into a new array.
ToString()	Returns the string representation of this read-only memory instance.
TryCopyTo(Memory<T>)	Tries to copy the contents of the readonly-only memory into the destination and returns a value that indicates whether the copy operation succeeded.

Operators

Expand table

Implicit(ArraySegment<T> to ReadOnlyMemory<T>)	Defines an implicit conversion of a ArraySegment<T> to a ReadOnlyMemory<T>.

Implicit(T[] to ReadOnlyMemory<T>)	Defines an implicit conversion of an array to a ReadOnlyMemory<T>.

Extension Methods

Expand table

Trim<T>(ReadOnlyMemory<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only memory region.
Trim<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
TrimEnd<T>(ReadOnlyMemory<T>, T)	Removes all trailing occurrences of a specified element from a read-only memory region.
TrimEnd<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only memory region.
TrimStart<T>(ReadOnlyMemory<T>, T)	Removes all leading occurrences of a specified element from a memory region.
TrimStart<T>(ReadOnlyMemory<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from a memory region.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Memory- and span-related types
- Memory<T> and Span<T> usage guidelines

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlyMemory<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ReadOnlyMemory<T>(T[])	Creates a new memory region over the entirety of the target array.
ReadOnlyMemory<T>(T[], Int32, Int32)	Creates a new memory region over the portion of the target array beginning at a specified position and including a specified number of elements.

ReadOnlyMemory<T>(T[])

Creates a new memory region over the entirety of the target array.

C#

```
public ReadOnlyMemory (T[]? array);
```

Parameters

array T[]

The target array.

Exceptions

ArrayTypeMismatchException

`array` is covariant, and the type of array is not exactly `T[]`.

Remarks

Returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ReadOnlyMemory<T>(T[], Int32, Int32)

Creates a new memory region over the portion of the target array beginning at a specified position and including a specified number of elements.

C#

```
public ReadOnlyMemory (T[]? array, int start, int length);
```

Parameters

`array` T[]

The target array.

`start` Int32

The index at which to begin the memory region.

`length` Int32

The number of items in the memory region.

Exceptions

ArrayTypeMismatchException

`array` is covariant, and the type of array is not exactly `T[]`.

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` identify a position outside the range of the array.

Remarks

Returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.Empty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an empty ReadOnlyMemory<T>.

C#

```
public static ReadOnlyMemory<T> Empty { get; }
```

Property Value

ReadOnlyMemory<T>

An empty read-only memory region.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.IsEmpty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether this read-only memory region is empty.

C#

```
public bool IsEmpty { get; }
```

Property Value

Boolean

`true` if the read-only memory region is empty (that is, its Length is 0); otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of items in the memory region.

C#

```
public int Length { get; }
```

Property Value

Int32

The number of items in the memory region.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.Span Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a span from the memory region.

C#

```
public ReadOnlySpan<T> Span { get; }
```

Property Value

ReadOnlySpan<T>

A memory span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.CopyTo(Memory<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of the read-only memory region into a destination memory region.

```
C#
public void CopyTo (Memory<T> destination);
```

Parameters

`destination` Memory<T>

The memory region to copy items into.

Exceptions

ArgumentException

`destination` is smaller than the source memory region.

Remarks

If the source and `destination` overlap, this method behaves as if the original values are in a temporary location before the destination is overwritten.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlyMemory<T>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Object)	Determines whether the specified object is equal to the current read-only memory region.
Equals(ReadOnlyMemory<T>)	Determines whether the current instance and a specified ReadOnlyMemory<T> objects are equal.

Equals(Object)

Determines whether the specified object is equal to the current read-only memory region.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare.

Returns

Boolean

`true` if the current instance and `obj` are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Equals(ReadOnlyMemory<T>)

Determines whether the current instance and a specified ReadOnlyMemory<T> objects are equal.

C#

```
public bool Equals (ReadOnlyMemory<T> other);
```

Parameters

`other` ReadOnlyMemory<T>

The read-only memory to compare.

Returns

Boolean

`true` if the current instance and `other` are equal; otherwise, `false`.

Implements

Equals(T)

Remarks

[!IMPORTANT]

Two ReadOnlyMemory<T> objects are equal if the memory regions point to the same array and have the same length. The method does not check to see if the contents are equal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this ReadOnlyMemory<T>.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.Pin Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a handle for the memory.

C#

```
public System.Buffers.MemoryHandle Pin ();
```

Returns

MemoryHandle

A handle for the memory.

Exceptions

ArgumentException

An instance with nonprimitive (non-blittable) members cannot be pinned.

Remarks

> **Important**
>
> The garbage collector will not move the memory until the returned **MemoryHandle** is disposed, which enables taking and using the memory's address.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlyMemory<T>.Slice Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Slice(Int32)	Forms a slice out of the given memory region, beginning at a specified position and continuing to its end.
Slice(Int32, Int32)	Forms a slice out of the given memory region starting at `start` position for `length` elements.

Slice(Int32)

Forms a slice out of the given memory region, beginning at a specified position and continuing to its end.

C#

```
public ReadOnlyMemory<T> Slice (int start);
```

Parameters

start Int32

The index at which to begin this slice.

Returns

ReadOnlyMemory<T>

A read-only memory region representing the desired slice.

Exceptions

ArgumentOutOfRangeException

`start` is less than 0 or greater than Length.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Slice(Int32, Int32)

Forms a slice out of the given memory region starting at `start` position for `length` elements.

```
C#

public ReadOnlyMemory<T> Slice (int start, int length);
```

Parameters

`start` Int32

The index at which to begin this slice.

`length` Int32

The desired length for the slice.

Returns

ReadOnlyMemory<T>

A read-only memory region representing the desired slice.

Exceptions

ArgumentOutOfRangeException

`start`, `length`, or `start` + `length` are outside the range of the current instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.ToArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents from the memory region into a new array.

```
C#
public T[] ToArray ();
```

Returns

T[]

An array.

Remarks

`ToArray` allocates on the heap and so should generally be avoided. However, it is sometimes necessary to bridge the gap with APIs written in terms of arrays.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

ReadOnlyMemory<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of this read-only memory instance.

C#

```
public override string ToString ();
```

Returns

String

For ReadOnlyMemory<T> with char elements, returns an instance of string that represents the characters pointed to by the memory. Otherwise, returns a String with the name of the type and the number of elements.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.TryCopyTo(Memory<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to copy the contents of the readonly-only memory into the destination and returns a value that indicates whether the copy operation succeeded.

```
C#
public bool TryCopyTo (Memory<T> destination);
```

Parameters

`destination` Memory<T>

The memory region to copy items into.

Returns

Boolean

`true` if the copy operation succeeded. `false` if `destination` is smaller than the current instance, and no data is written to `destination`.

Remarks

If the source and `destination` overlap, this method behaves as if the original values are in a temporary location before the destination is overwritten.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlyMemory<T>.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(ArraySegment<T> to ReadOnlyMemory<T>)	Defines an implicit conversion of a ArraySegment<T> to a ReadOnlyMemory<T>.
Implicit(T[] to ReadOnlyMemory<T>)	Defines an implicit conversion of an array to a ReadOnlyMemory<T>.

Implicit(ArraySegment<T> to ReadOnlyMemory<T>)

Defines an implicit conversion of a ArraySegment<T> to a ReadOnlyMemory<T>.

C#

```
public static implicit operator ReadOnlyMemory<T> (ArraySegment<T>
segment);
```

Parameters

`segment` ArraySegment<T>

The array segment to convert.

Returns

ReadOnlyMemory<T>

A read-only memory representation of the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(T[] to ReadOnlyMemory<T>)

Defines an implicit conversion of an array to a ReadOnlyMemory<T>.

C#

```
public static implicit operator ReadOnlyMemory<T> (T[]? array);
```

Parameters

`array` T[]

An array of type T to convert.

Returns

ReadOnlyMemory<T>

The read-only memory representation of the array of type T.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Enumerator Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an enumerator for the elements of a ReadOnlySpan<T>.

```
C#
public ref struct ReadOnlySpan<T>.Enumerator
```

Type Parameters

T

Inheritance Object → ValueType → ReadOnlySpan<T>.Enumerator

Remarks

The C# foreach of the C# language and the For Each...Next construct in Visual Basic hides the complexity of enumerators. Instead of directly manipulating the enumerator, using `foreach` or `For Each...Next` is recommended.

Initially, the enumerator is positioned before the first element in the ReadOnlySpan<T>. At this position, Current is undefined. You must call MoveNext to advance the enumerator to the first item in the ReadOnlySpan<T> before reading the value of Current.

Current returns the same value until MoveNext is called. MoveNext sets Current to the next item in the ReadOnlySpan<T>.

If MoveNext passes the end of the ReadOnlySpan<T>, MoveNext returns `false`. When the enumerator is at this state, subsequent calls to MoveNext also return `false` and Current is undefined. You cannot set Current to the first item in the ReadOnlySpan<T> again; you must create a new enumerator instance instead.

Though the ReadOnlySpan<T> is allocated on the stack, the underlying data on which the ReadOnlySpan<T> points to, may not be. Therefore, enumerating through a ReadOnlySpan<T> is intrinsically not a thread-safe procedure. To guarantee thread safety during enumeration, you must implement your own synchronization.

Unlike some other enumerator structures in .NET, the ReadOnlySpan<T>.Enumerator:

- Does not implement the IEnumerator or IEnumerator<T> interface. This is because ReadOnlySpan<T>.Enumerator is a ref struct and cannot be boxed.
- Does not include a `Reset` method, which can set the enumerator to its initial position before the first element in the span. (The IEnumerator.Reset() method must be implemented as part of the interface, but most implementors either throw an exception or provide no implementation.)

Properties

Expand table

Current	Gets a reference to the item at the current position of the enumerator.

Methods

Expand table

MoveNext()	Advances the enumerator to the next item of the ReadOnlySpan<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.Enumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a reference to the item at the current position of the enumerator.

```
C#
public ref readonly T Current { get; }
```

Property Value

T

The element in the ReadOnlySpan<T> at the current position of the enumerator.

Remarks

`Current` is undefined under either of the following conditions:

- Immediately after the enumerator is created, the enumerator is positioned before the first element in the span. MoveNext must be called to advance the enumerator to the first element of the span before reading the value of `Current`.
- The last call to MoveNext returned `false`, which indicates the end of the span.

`Current` returns the same value until MoveNext is called. MoveNext sets `Current` to the next item in the span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.Enumerator.MoveNext Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Advances the enumerator to the next item of the ReadOnlySpan<T>.

```
C#
public bool MoveNext ();
```

Returns

Boolean

`true` if the enumerator successfully advanced to the next item; `false` if the end of the span has been passed.

Remarks

After an enumerator is created, it is positioned before the first element in the span, and the first call to `MoveNext` advances the enumerator to the first item in the span.

If `MoveNext` passes the end of the span, `MoveNext` returns `false`. When the enumerator is at this state, subsequent calls to `MoveNext` also return `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a type-safe and memory-safe read-only representation of a contiguous region of arbitrary memory.

C#

```
[System.Runtime.InteropServices.Marshalling.NativeMarshalling(typeof(System.
Runtime.InteropServices.Marshalling.ReadOnlySpanMarshaller<,>))]
public readonly ref struct ReadOnlySpan<T>
```

Type Parameters

T

The type of items in the ReadOnlySpan<T>.

Inheritance Object → ValueType → ReadOnlySpan<T>

Attributes NativeMarshallingAttribute

Remarks

`ReadOnlySpan<T>` is a ref struct that is allocated on the stack and can never escape to the managed heap. Ref struct types have a number of restrictions to ensure that they cannot be promoted to the managed heap, including that they can't be boxed, captured in lambda expressions, assigned to variables of type Object, assigned to `dynamic` variables, and they cannot implement any interface type.

A `ReadOnlySpan<T>` instance is often used to reference the elements of an array or a portion of an array. Unlike an array, however, a `ReadOnlySpan<T>` instance can point to managed memory, native memory, or memory managed on the stack.

Constructors

Expand table

ReadOnlySpan<T>(T)	Creates a new ReadOnlySpan<T> of length 1 around the specified reference.
ReadOnlySpan<T>(T[])	Creates a new ReadOnlySpan<T> over the entirety of a specified array.
ReadOnlySpan<T>(T[], Int32, Int32)	Creates a new ReadOnlySpan<T> that includes a specified number of elements of an array starting at a specified index.
ReadOnlySpan<T>(Void*, Int32)	Creates a new ReadOnlySpan<T> from a specified number of `T` elements starting at a specified memory address.

Properties

Expand table

Empty	Returns an empty ReadOnlySpan<T>.
IsEmpty	Returns a value that indicates the current read-only span is empty.
Item[Int32]	Gets an item from the read-only span at the specified zero-based index.
Length	The number of items in the read-only span.

Methods

Expand table

CopyTo(Span<T>)	Copies the contents of this ReadOnlySpan<T> into a destination Span<T>.
Equals(Object)	**Obsolete.** Not supported. Throws a NotSupportedException.
GetEnumerator()	Returns an enumerator for this ReadOnlySpan<T>.
GetHashCode()	**Obsolete.** Not supported. Throws a NotSupportedException.
GetPinnable Reference()	Returns a read-only reference to an object of type T that can be used for pinning. This method is intended to support .NET compilers and is not intended to be called by user code.

Slice(Int32)	Forms a slice out of the current read-only span that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current read-only span starting at a specified index for a specified length.
ToArray()	Copies the contents of this read-only span into a new array.
ToString()	Returns the string representation of this ReadOnlySpan<T>.
TryCopyTo(Span<T>)	Attempts to copy the contents of this ReadOnlySpan<T> into a Span<T> and returns a value to indicate whether or not the operation succeeded.

Operators

Expand table

Equality(ReadOnlySpan<T>, ReadOnlySpan<T>)	Returns a value that indicates whether two ReadOnlySpan<T> instances are equal.
Implicit(ArraySegment<T> to ReadOnlySpan<T>)	Defines an implicit conversion of an ArraySegment<T> to a ReadOnlySpan<T>.
Implicit(T[] to ReadOnlySpan<T>)	Defines an implicit conversion of an array to a ReadOnlySpan<T>.
Inequality(ReadOnlySpan<T>, ReadOnlySpan<T>)	Returns a value that indicates whether two ReadOnlySpan<T> instances are not equal.

Extension Methods

Expand table

ToImmutableArray<T>(ReadOnlySpan<T>)	Produce an immutable array of contents from specified elements.
BinarySearch<T>(ReadOnlySpan<T>, IComparable<T>)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified IComparable<T> generic interface.
BinarySearch<T,TComparer>(ReadOnlySpan<T>, T, TComparer)	Searches an entire sorted ReadOnlySpan<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T,TComparable>(ReadOnlySpan<T>, TComparable)	Searches an entire sorted ReadOnlySpan<T> for a value using the specified `TComparable` generic

	type.
CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines the length of any common prefix shared between `span` and `other`.
Contains<T>(ReadOnlySpan<T>, T)	Indicates whether a specified value is found in a read-only span. Values are compared using IEquatable{T}.Equals(T).
ContainsAny<T>(ReadOnlySpan<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for any value other than the specified `value`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
ContainsAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.

Count<T>(ReadOnlySpan<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Counts the number of times `value` occurs in `span`.
EndsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a read-only span.
IndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.

IndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOf<T>(ReadOnlySpan<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOf<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExceptInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyInRange<T>(ReadOnlySpan<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.

Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences overlap in memory.
Overlaps<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, Int32)	Determines whether two read-only sequences overlap in memory and outputs the element offset.
Replace<T>(ReadOnlySpan<T>, Span<T>, T, T)	Copies `source` to `destination`, replacing all occurrences of `oldValue` with `newValue`.
SequenceCompareTo<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines the relative order of two read-only sequences by comparing their elements using IComparable{T}.CompareTo(T).
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether two read-only sequences are equal by comparing the elements using IEquatable{T}.Equals(T).
SequenceEqual<T>(ReadOnlySpan<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
StartsWith<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Determines whether a specified sequence appears at the start of a read-only span.
Trim<T>(ReadOnlySpan<T>, T)	Removes all leading and trailing occurrences of a specified element from a read-only span.
Trim<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a read-only span.
TrimEnd<T>(ReadOnlySpan<T>, T)	Removes all trailing occurrences of a specified element from a read-only span.
TrimEnd<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a read-only span.
TrimStart<T>(ReadOnlySpan<T>, T)	Removes all leading occurrences of a specified element from the span.
TrimStart<T>(ReadOnlySpan<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from the span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Memory- and span-related types
- Memory<T> and Span<T> usage guidelines

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ReadOnlySpan<T>(T)	Creates a new ReadOnlySpan<T> of length 1 around the specified reference.
ReadOnlySpan<T>(T[])	Creates a new ReadOnlySpan<T> over the entirety of a specified array.
ReadOnlySpan<T>(Void*, Int32)	Creates a new ReadOnlySpan<T> from a specified number of `T` elements starting at a specified memory address.
ReadOnlySpan<T>(T[], Int32, Int32)	Creates a new ReadOnlySpan<T> that includes a specified number of elements of an array starting at a specified index.

ReadOnlySpan<T>(T)

Creates a new ReadOnlySpan<T> of length 1 around the specified reference.

C#

```
public ReadOnlySpan (ref T reference);
```

Parameters

`reference` T

A reference to data.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ReadOnlySpan<T>(T[])

Creates a new ReadOnlySpan<T> over the entirety of a specified array.

```
C#
public ReadOnlySpan (T[]? array);
```

Parameters

`array` T[]

The array from which to create the ReadOnlySpan<T>.

Remarks

If the array is null, this constructor returns a null `ReadOnlySpan<T>`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ReadOnlySpan<T>(Void*, Int32)

> Important
>
> This API is not CLS-compliant.

Creates a new ReadOnlySpan<T> from a specified number of `T` elements starting at a specified memory address.

```C#
[System.CLSCompliant(false)]
public ReadOnlySpan (void* pointer, int length);
```

Parameters

`pointer` Void*

A pointer to the starting address of a specified number of `T` elements in memory.

`length` Int32

The number of `T` elements to be included in the ReadOnlySpan<T>.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`T` is a reference type or contains references and therefore cannot be stored in unmanaged memory.

ArgumentOutOfRangeException

`length` is negative.

Remarks

This constructor should be used with care, since it creates arbitrarily typed `T`s from a block of memory.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

ReadOnlySpan<T>(T[], Int32, Int32)

Creates a new ReadOnlySpan<T> that includes a specified number of elements of an array starting at a specified index.

```
C#

public ReadOnlySpan (T[]? array, int start, int length);
```

Parameters

array T[]

The array from which to create the ReadOnlySpan<T>.

start Int32

The index of the first element to include.

length Int32

The number of elements to include.

Exceptions

ArgumentOutOfRangeException

`array` is `null`, but `start` or `length` is non-zero.

-or-

`start` is outside the bounds of the array.

-or-

`start` and `length` exceeds the number of elements in the array.

Remarks

This method returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Empty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an empty ReadOnlySpan<T>.

C#

```
public static ReadOnlySpan<T> Empty { get; }
```

Property Value

ReadOnlySpan<T>

An empty ReadOnlySpan<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.IsEmpty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates the current read-only span is empty.

C#

```
public bool IsEmpty { get; }
```

Property Value

Boolean

`true` when the current span is empty; otherwise `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an item from the read-only span at the specified zero-based index.

C#

```
public ref readonly T this[int index] { get; }
```

Parameters

`index` Int32

The zero-based index of the element to get.

Property Value

T

The element from the read-only span.

Exceptions

IndexOutOfRangeException

`index` is less then zero or greater than or equal to Length.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The number of items in the read-only span.

C#

```
public int Length { get; }
```

Property Value

Int32

The number of items.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.CopyTo(Span<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this ReadOnlySpan<T> into a destination Span<T>.

```
C#
public void CopyTo (Span<T> destination);
```

Parameters

destination Span<T>

The span to copy items into.

Exceptions

ArgumentException

`destination` is shorter than the source ReadOnlySpan<T>.

Remarks

If the `source` and `destination` overlap, the entirety of `source` is handled as if it was copied to a temporary location before it is copied to `destination`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Equals() on ReadOnlySpan will always throw an exception. Use the equality operator instead.

Not supported. Throws a NotSupportedException.

C#

```
[System.Obsolete("Equals() on ReadOnlySpan will always throw an exception.
Use the equality operator instead.")]
public override bool Equals (object? obj);
```

Parameters

obj Object

Not supported.

Returns

Boolean

Not supported.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Always thrown by this method.

Remarks

Not supported.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Standard	*(2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator for this ReadOnlySpan<T>.

C#

```
public ReadOnlySpan<T>.Enumerator GetEnumerator ();
```

Returns

ReadOnlySpan<T>.Enumerator

An enumerator for this read-only span.

Remarks

Instead of calling the GetEnumerator method directly, you can use the C# `foreach` statement and the Visual Basic `For Each`...`Next` construct to enumerate a ReadOnlySpan<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> GetHashCode() on ReadOnlySpan will always throw an exception.

Not supported. Throws a NotSupportedException.

C#

```
[System.Obsolete("GetHashCode() on ReadOnlySpan will always throw an exception.")]
public override int GetHashCode ();
```

Returns

Int32

Not supported.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Always thrown by this method.

Remarks

Not supported.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Standard	*(2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.GetPinnableReference Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a read-only reference to an object of type T that can be used for pinning.

This method is intended to support .NET compilers and is not intended to be called by user code.

```
C#

public ref readonly T GetPinnableReference ();
```

Returns

T

A reference to the element of the span at index 0, or `null` if the span is empty.

Examples

The following example demonstrates creating an integer array, pinning it, and writing each element to the console.

```
C#

using System;

// Note: you must compile this sample using the unsafe flag.
// From the command line, type the following: csc sample.cs /unsafe

public class Example
{
    public static unsafe void Main()
    {
        int[] array = CreateInt32Array();

        // Create a span, pin it, and print its elements.
```

```
        Span<int> span = array.AsSpan();
        fixed (int* spanPtr = span)
        {
            Console.WriteLine($"Span contains {span.Length} elements:");
            for (int i = 0; i < span.Length; i++)
            {
                Console.WriteLine(spanPtr[i]);
            }
            Console.WriteLine();
        }

        // Create a read-only span, pin it, and print its elements.
        ReadOnlySpan<int> readonlyspan = array.AsSpan();
        fixed (int* readonlyspanPtr = readonlyspan)
        {
            Console.WriteLine($"ReadOnlySpan contains {readonlyspan.Length}
elements:");
            for (int i = 0; i < readonlyspan.Length; i++)
            {
                Console.WriteLine(readonlyspanPtr[i]);
            }
            Console.WriteLine();
        }
    }

    private static int[] CreateInt32Array()
    {
        return new int[] { 100, 200, 300, 400, 500 };
    }
}

// The example displays the following output:
//       Span contains 5 elements:
//       100
//       200
//       300
//       400
//       500
//
//       ReadOnlySpan contains 5 elements:
//       100
//       200
//       300
//       400
//       500
```

Remarks

Applications should not directly call `GetPinnableReference`. Instead, callers should use their language's normal pinning syntax, such as C#'s `fixed` statement.

If pinning a `ReadOnlySpan<char>`, the resulting `char*` **is not** assumed to be null-terminated. This behavior is different from pinning a `string`, where the resulting `char*` is guaranteed to be null-terminated.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.Slice Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Slice(Int32)	Forms a slice out of the current read-only span that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current read-only span starting at a specified index for a specified length.

Slice(Int32)

Forms a slice out of the current read-only span that begins at a specified index.

C#

```
public ReadOnlySpan<T> Slice (int start);
```

Parameters

`start` Int32

The index at which to begin the slice.

Returns

ReadOnlySpan<T>

A span that consists of all elements of the current span from `start` to the end of the span.

Exceptions

ArgumentOutOfRangeException

`start` is greater than the number of items in the read-only span.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Slice(Int32, Int32)

Forms a slice out of the current read-only span starting at a specified index for a specified length.

```
C#

public ReadOnlySpan<T> Slice (int start, int length);
```

Parameters

`start` Int32

The index at which to begin this slice.

`length` Int32

The desired length for the slice.

Returns

ReadOnlySpan<T>

A read-only span that consists of `length` elements from the current span starting at `start`.

Exceptions

ArgumentOutOfRangeException

`start` is less than zero or greater than Length.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.ToArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this read-only span into a new array.

C#

```
public T[] ToArray ();
```

Returns

T[]

An array containing the data in the current span.

Remarks

This method performs a heap allocation and data copy, and therefore, should be avoided if possible. It it sometimes necessary when APIs that support ReadOnlySpan<T> are not available, but APIs that work with arrays are.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of this ReadOnlySpan<T>.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this instance.

Remarks

For a `ReadOnlySpan<Char>`, the `ToString` method returns a String that contains the characters pointed to by the span. Otherwise, it returns a String with the name of the type and the number of elements that the span contains.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.TryCopyTo(Span<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Attempts to copy the contents of this ReadOnlySpan<T> into a Span<T> and returns a value to indicate whether or not the operation succeeded.

```
C#

public bool TryCopyTo (Span<T> destination);
```

Parameters

destination Span<T>

The target of the copy operation.

Returns

Boolean

`true` if the copy operation succeeded; otherwise, `false`.

Remarks

If the `source` and `destination` overlap, the entirety of `source` is handled as if it was copied to a temporary location before it is copied to `destination`.

If the `destination` length is shorter than `source`, no items are copied and the method returns `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ReadOnlySpan<T>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two ReadOnlySpan<T> instances are equal.

```
C#

public static bool operator == (ReadOnlySpan<T> left, ReadOnlySpan<T>
right);
```

Parameters

left ReadOnlySpan<T>

The first span to compare.

right ReadOnlySpan<T>

The second span to compare.

Returns

Boolean

`true` if the two ReadOnlySpan<T> instances are equal; otherwise, `false`.

Remarks

This tests if two ReadOnlySpan<T> instances point to the same starting memory location, and have the same Length values. This does not compare the contents of two ReadOnlySpan<T> instances.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(ArraySegment<T> to ReadOnlySpan<T>)	Defines an implicit conversion of an ArraySegment<T> to a ReadOnlySpan<T>.
Implicit(T[] to ReadOnlySpan<T>)	Defines an implicit conversion of an array to a ReadOnlySpan<T>.

Implicit(ArraySegment<T> to ReadOnlySpan<T>)

Defines an implicit conversion of an ArraySegment<T> to a ReadOnlySpan<T>.

C#

```
public static implicit operator ReadOnlySpan<T> (ArraySegment<T>
segment);
```

Parameters

`segment` ArraySegment<T>

The array segment to be converted to a ReadOnlySpan<T>.

Returns

ReadOnlySpan<T>

A read-only span that corresponds to the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(T[] to ReadOnlySpan<T>)

Defines an implicit conversion of an array to a ReadOnlySpan<T>.

C#

```
public static implicit operator ReadOnlySpan<T> (T[]? array);
```

Parameters

`array` T[]

The array to convert to a ReadOnlySpan<T>.

Returns

ReadOnlySpan<T>

The read-only span that corresponds to `array`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ReadOnlySpan<T>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two ReadOnlySpan<T> instances are not equal.

```
C#

public static bool operator != (ReadOnlySpan<T> left, ReadOnlySpan<T>
right);
```

Parameters

left ReadOnlySpan<T>

The first read-only span to compare.

right ReadOnlySpan<T>

The second read-only span to compare.

Returns

Boolean

`true` if the two ReadOnlySpan<T> instances are not equal; otherwise, `false`.

Remarks

Two ReadOnlySpan<T> instances are not equal if they have different lengths or if the corresponding elements of `left` and `right` point to different memory locations.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ResolveEventArgs Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides data for loader resolution events, such as the TypeResolve, ResourceResolve, ReflectionOnlyAssemblyResolve, and AssemblyResolve events.

C#

```
public class ResolveEventArgs : EventArgs
```

Inheritance Object → EventArgs → ResolveEventArgs

Remarks

When the loader cannot resolve an assembly reference and a handler has been provided for the appropriate loader resolution event, the event is raised and the ResolveEventArgs contains information about the item to be resolved.

- The Name property contains the name of the item to be resolved.
- Beginning with the .NET Framework 4, the RequestingAssembly property contains the assembly that requested an assembly that can provide the named item. For more information, see the RequestingAssembly property.

Constructors

Expand table

ResolveEventArgs(String)	Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve.
ResolveEventArgs(String, Assembly)	Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve and the assembly whose dependency is being resolved.

Properties

Expand table

Name	Gets the name of the item to resolve.
RequestingAssembly	Gets the assembly whose dependency is being resolved.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeResolve
- ResourceResolve
- ReflectionOnlyAssemblyResolve
- AssemblyResolve

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ResolveEventArgs Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ResolveEventArgs class.

Overloads

Expand table

ResolveEventArgs(String)	Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve.
ResolveEventArgs(String, Assembly)	Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve and the assembly whose dependency is being resolved.

ResolveEventArgs(String)

Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve.

C#

```
public ResolveEventArgs (string name);
```

Parameters

`name` String

The name of an item to resolve.

Remarks

This constructor is typically called only by the common language runtime.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ResolveEventArgs(String, Assembly)

Initializes a new instance of the ResolveEventArgs class, specifying the name of the item to resolve and the assembly whose dependency is being resolved.

C#

```
public ResolveEventArgs (string name, System.Reflection.Assembly?
requestingAssembly);
```

Parameters

`name` String

The name of an item to resolve.

`requestingAssembly` Assembly

The assembly whose dependency is being resolved.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ResolveEventArgs.Name Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the name of the item to resolve.

```
C#
public string Name { get; }
```

Property Value

String

The name of the item to resolve.

Remarks

For the AssemblyResolve event, Name is the assembly name before policy is applied.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ResolveEventArgs.RequestingAssembly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the assembly whose dependency is being resolved.

C#

```
public System.Reflection.Assembly? RequestingAssembly { get; }
```

Property Value

Assembly

The assembly that requested the item specified by the Name property.

Remarks

The assembly that is returned by this property is an assembly that was unable to resolve the item specified by the Name property, because the item did not exist in that assembly, in any of its loaded dependencies, or in any dependencies the loader could find through probing.

For example, suppose the current assembly uses the Assembly.LoadFile method to load assembly A from a directory outside the probing path, in order to use class CA. Suppose further that class CA uses class CB, in assembly B, and that assembly A has a compile-time reference to assembly B, also located outside the probing path. When the current assembly attempts to instantiate CA, the loader attempts to resolve the reference to assembly B. However, because assembly A was loaded by using the Assembly.LoadFile method, the loader cannot resolve the dependency. If there is a handler for the AppDomain.AssemblyResolve event, the event is raised, and the RequestingAssembly property of the resulting ResolveEventArgs object contains assembly A, because assembly A is the assembly that is missing a dependency.

The semantics of the RequestingAssembly property vary, depending on how the requesting assembly was loaded:

- Load context: The value of the property might be `null`. The load context is load-order independent, and the identity of the requesting assembly is not necessarily meaningful. A non-null example occurs when the Type.GetType method is called and the specified type is not known to the requesting assembly or any of its dependencies.
- Load-from context: The property can be used as a probing hint, but the event handler should not get more than one request for a given assembly name.
- No context (that is, the requesting assembly was loaded by using the Assembly.LoadFile method, or as a byte stream, or in mixed mode): The handler can use the RequestingAssembly property to load different implementations of the same assembly based on the identity of the requesting assembly.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeResolve
- ResourceResolve
- ReflectionOnlyAssemblyResolve
- AssemblyResolve

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ResolveEventHandler Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a method that handles the TypeResolve, ResourceResolve, or AssemblyResolve event of an AppDomain.

```
C#

public delegate System.Reflection.Assembly? ResolveEventHandler(object?
sender, ResolveEventArgs args);
```

Parameters

sender Object

The source of the event.

args ResolveEventArgs

The event data.

Return Value

Assembly

The assembly that resolves the type, assembly, or resource; or `null` if the assembly cannot be resolved.

Remarks

If the runtime class loader cannot resolve a reference to an assembly, type, or resource, the corresponding events are raised to give the callback a chance to tell the runtime which assembly the referenced assembly, type, or resource is in. It is the responsibility of the ResolveEventHandler to return the assembly that resolves the type, assembly, or resource, or to return null if the assembly is not recognized. For more information, see Resolving Assembly Loads and the AppDomain.ResourceResolve, AppDomain.AssemblyResolve, and AppDomain.ReflectionOnlyAssemblyResolve events.

> **Important**
>
> Beginning with the .NET Framework 4, the **System.ResolveEventHandler** event is raised for all assemblies, including resource assemblies. In earlier versions, the event was not raised for resource assemblies. If the operating system is localized, the handler might be called multiple times: once for each culture in the fallback chain.

Every derived class of Delegate and MulticastDelegate has a constructor and an `Invoke` method. See the C++ code example in the description for the Delegate class.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ResourceResolve
- AssemblyResolve
- ReflectionOnlyAssemblyResolve
- Resolve assembly loads

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

RuntimeArgumentHandle Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

References a variable-length argument list.

C#

```
public ref struct RuntimeArgumentHandle
```

Inheritance Object → ValueType → RuntimeArgumentHandle

Remarks

This structure has no members other than those that it inherits from the Object and ValueType classes. It exists only to support C/C++ programming language functions that take a variable number of parameters. An instance of the structure is passed to the ArgIterator constructor.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

RuntimeFieldHandle Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a field using an internal metadata token.

C#

```
public struct RuntimeFieldHandle : IEquatable<RuntimeFieldHandle>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → RuntimeFieldHandle

Implements ISerializable , IEquatable<RuntimeFieldHandle>

Remarks

The value of an instance of the RuntimeFieldHandle structure is a handle to the field represented by the instance.

Properties

Expand table

Value	Gets a handle to the field represented by the current instance.

Methods

Expand table

Equals(Object)	Indicates whether the current instance is equal to the specified object.
Equals(RuntimeFieldHandle)	Indicates whether the current instance is equal to the

	specified RuntimeFieldHandle.
FromIntPtr(IntPtr)	Returns a new RuntimeFieldHandle object created from a handle to a RuntimeFieldInfo.
GetHashCode()	Returns the hash code for this instance.
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo with the data necessary to deserialize the field represented by the current instance.
ToIntPtr(RuntimeFieldHandle)	Returns the internal pointer representation of a RuntimeFieldHandle object.

Operators

Expand table

Equality(RuntimeFieldHandle, RuntimeFieldHandle)	Indicates whether two RuntimeFieldHandle structures are equal.
Inequality(RuntimeFieldHandle, RuntimeFieldHandle)	Indicates whether two RuntimeFieldHandle structures are not equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeFieldHandle.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a handle to the field represented by the current instance.

C#

```
public IntPtr Value { get; }
```

Property Value

IntPtr

An IntPtr that contains the handle to the field represented by the current instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeFieldHandle.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether an object represents the same field as the current RuntimeFieldHandle.

Overloads

Expand table

Equals(Object)	Indicates whether the current instance is equal to the specified object.
Equals(RuntimeFieldHandle)	Indicates whether the current instance is equal to the specified RuntimeFieldHandle.

Equals(Object)

Indicates whether the current instance is equal to the specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare to the current instance.

Returns

Boolean

`true` if `obj` is a RuntimeFieldHandle and equal to the value of the current instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(RuntimeFieldHandle)

Indicates whether the current instance is equal to the specified RuntimeFieldHandle.

C#

```
public bool Equals (RuntimeFieldHandle handle);
```

Parameters

`handle` RuntimeFieldHandle

The RuntimeFieldHandle to compare to the current instance.

Returns

Boolean

`true` if the value of `handle` is equal to the value of the current instance; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeFieldHandle.FromIntPtr(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new RuntimeFieldHandle object created from a handle to a RuntimeFieldInfo.

C#

```
public static RuntimeFieldHandle FromIntPtr (IntPtr value);
```

Parameters

`value` IntPtr

An IntPtr handle to a RuntimeFieldInfo to create a RuntimeFieldHandle object from.

Returns

RuntimeFieldHandle

A new RuntimeFieldHandle object that corresponds to the value parameter.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeFieldHandle.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that is the hash code for this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetHashCode()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

RuntimeFieldHandle.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo with the data necessary to deserialize the field represented by the current instance.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo object to populate with serialization information.

`context` StreamingContext

(Reserved) The place to store and retrieve serialized data.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

SerializationException

The Value property of the current instance is not a valid handle.

Remarks

The `context` parameter is reserved; it does not currently participate in this operation.

For more information, see the SerializationInfo.AddValue method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeFieldHandle.ToIntPtr(RuntimeFieldHandle) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the internal pointer representation of a RuntimeFieldHandle object.

C#

```
public static IntPtr ToIntPtr (RuntimeFieldHandle value);
```

Parameters

`value` RuntimeFieldHandle

A RuntimeFieldHandle object to retrieve an internal pointer representation from.

Returns

IntPtr

An IntPtr object that represents a RuntimeFieldHandle object.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeFieldHandle.Equality(RuntimeFieldHandle, RuntimeFieldHandle) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two RuntimeFieldHandle structures are equal.

C#

```
public static bool operator == (RuntimeFieldHandle left, RuntimeFieldHandle right);
```

Parameters

`left` RuntimeFieldHandle

The RuntimeFieldHandle to compare to `right`.

`right` RuntimeFieldHandle

The RuntimeFieldHandle to compare to `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Remarks

The equivalent method for this operator is RuntimeFieldHandle.Equals(RuntimeFieldHandle).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeFieldHandle.Inequality(RuntimeFieldHandle, RuntimeFieldHandle) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two RuntimeFieldHandle structures are not equal.

C#

```
public static bool operator != (RuntimeFieldHandle left, RuntimeFieldHandle right);
```

Parameters

`left` RuntimeFieldHandle

The RuntimeFieldHandle to compare to `right`.

`right` RuntimeFieldHandle

The RuntimeFieldHandle to compare to `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Remarks

The equivalent method for this operator is RuntimeFieldHandle.Equals(RuntimeFieldHandle).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

RuntimeMethodHandle is a handle to the internal metadata representation of a method.

C#

```
public struct RuntimeMethodHandle : IEquatable<RuntimeMethodHandle>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → RuntimeMethodHandle

Implements ISerializable , IEquatable<RuntimeMethodHandle>

Properties

Expand table

Value	Gets the value of this instance.

Methods

Expand table

Equals(Object)	Indicates whether this instance is equal to a specified object.
Equals(RuntimeMethodHandle)	Indicates whether this instance is equal to a specified RuntimeMethodHandle.
FromIntPtr(IntPtr)	Returns a new RuntimeMethodHandle object created from a handle to a RuntimeMethodInfo.
GetFunctionPointer()	Obtains a pointer to the method represented by this instance.

GetHashCode()	Returns the hash code for this instance.
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo with the data necessary to deserialize the field represented by this instance.
ToIntPtr(RuntimeMethodHandle)	Returns the internal pointer representation of a RuntimeMethodHandle object.

Operators

Expand table

Equality(RuntimeMethodHandle, RuntimeMethodHandle)	Indicates whether two instances of RuntimeMethodHandle are equal.
Inequality(RuntimeMethodHandle, RuntimeMethodHandle)	Indicates whether two instances of RuntimeMethodHandle are not equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of this instance.

C#

```
public IntPtr Value { get; }
```

Property Value

IntPtr

A RuntimeMethodHandle that is the internal metadata representation of a method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether one RuntimeMethodHandle is equal to another.

Overloads

Expand table

Equals(Object)	Indicates whether this instance is equal to a specified object.
Equals(RuntimeMethodHandle)	Indicates whether this instance is equal to a specified RuntimeMethodHandle.

Equals(Object)

Indicates whether this instance is equal to a specified object.

```
C#
public override bool Equals (object? obj);
```

Parameters

obj Object

A Object to compare to this instance.

Returns

Boolean

`true` if `obj` is a RuntimeMethodHandle and equal to the value of this instance; otherwise, `false`.

See also

- Value

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(RuntimeMethodHandle)

Indicates whether this instance is equal to a specified RuntimeMethodHandle.

C#

```
public bool Equals (RuntimeMethodHandle handle);
```

Parameters

`handle` RuntimeMethodHandle

A RuntimeMethodHandle to compare to this instance.

Returns

Boolean

`true` if `handle` is equal to the value of this instance; otherwise, `false`.

Implements

Equals(T)

See also

- Value

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.FromIntPtr(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new RuntimeMethodHandle object created from a handle to a RuntimeMethodInfo.

C#

```
public static RuntimeMethodHandle FromIntPtr (IntPtr value);
```

Parameters

`value` IntPtr

An IntPtr handle to a RuntimeMethodInfo to create a RuntimeMethodHandle object from.

Returns

RuntimeMethodHandle

A new RuntimeMethodHandle object that corresponds to the value parameter.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.GetFunctionPointer Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Obtains a pointer to the method represented by this instance.

```
C#
public IntPtr GetFunctionPointer ();
```

Returns

IntPtr

A pointer to the method represented by this instance.

Exceptions

SecurityException

The caller does not have the necessary permission to perform this operation.

Remarks

If the method has the System.Runtime.InteropServices.UnmanagedCallersOnlyAttribute attribute, then the returned value is an unmanaged function pointer with the same calling convention as specified in the attribute. If the method this handle represents is static, then the returned value can be cast to a C# managed function pointer type with the same signature. For instance method handles, the value is not easily usable from user code and is meant exclusively for usage within the runtime.

For methods that don't have the System.Runtime.InteropServices.UnmanagedCallersOnlyAttribute, use Marshal.GetFunctionPointerForDelegate to get a function pointer that can be passed to native code.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeMethodHandle.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo with the data necessary to deserialize the field represented by this instance.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with serialization information.

`context` StreamingContext

(Reserved) The place to store and retrieve serialized data.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

SerializationException

Value is invalid.

Remarks

The `context` parameter is reserved; it does not currently participate in this operation.

For more information, see the SerializationInfo.AddValue method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.ToIntPtr(RuntimeMethodHandle) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the internal pointer representation of a RuntimeMethodHandle object.

C#

```
public static IntPtr ToIntPtr (RuntimeMethodHandle value);
```

Parameters

`value` RuntimeMethodHandle

A RuntimeMethodHandle object to retrieve an internal pointer representation from.

Returns

IntPtr

An IntPtr object that represents a RuntimeMethodHandle object.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeMethodHandle.Equality(RuntimeMethodHandle, RuntimeMethodHandle) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two instances of RuntimeMethodHandle are equal.

C#

```
public static bool operator == (RuntimeMethodHandle left,
RuntimeMethodHandle right);
```

Parameters

`left` RuntimeMethodHandle

A RuntimeMethodHandle to compare to `right`.

`right` RuntimeMethodHandle

A RuntimeMethodHandle to compare to `left`.

Returns

Boolean

`true` if the value of `left` is equal to the value of `right`; otherwise, `false`.

Remarks

The equivalent method for this operator is RuntimeMethodHandle.Equals(RuntimeMethodHandle).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeMethodHandle.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two instances of RuntimeMethodHandle are not equal.

C#

```
public static bool operator != (RuntimeMethodHandle left,
RuntimeMethodHandle right);
```

Parameters

`left` RuntimeMethodHandle

A RuntimeMethodHandle to compare to `right`.

`right` RuntimeMethodHandle

A RuntimeMethodHandle to compare to `left`.

Returns

Boolean

`true` if the value of `left` is unequal to the value of `right`; otherwise, `false`.

Remarks

The equivalent method for this operator is RuntimeMethodHandle.Equals(RuntimeMethodHandle).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a type using an internal metadata token.

C#

```
public struct RuntimeTypeHandle : IEquatable<RuntimeTypeHandle>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → RuntimeTypeHandle

Implements ISerializable , IEquatable<RuntimeTypeHandle>

Examples

The following example demonstrates how to obtain a RuntimeTypeHandle from a type or from an object, and how to turn the handle back into a type.

C#

```
using System;
using System.Reflection;

public class MyClass1
{
    private int x=0;
    public int MyMethod()
    {
        return x;
    }
}

public class MyClass2
{
    public static void Main()
    {
        MyClass1 myClass1 = new MyClass1();
```

```
        // Get the RuntimeTypeHandle from an object.
        RuntimeTypeHandle myRTHFromObject = Type.GetTypeHandle(myClass1);
        // Get the RuntimeTypeHandle from a type.
        RuntimeTypeHandle myRTHFromType = typeof(MyClass1).TypeHandle;

        Console.WriteLine("\nmyRTHFromObject.Value:  {0}",
myRTHFromObject.Value);
        Console.WriteLine("myRTHFromObject.GetType():  {0}",
myRTHFromObject.GetType());
        Console.WriteLine("Get the type back from the handle...");
        Console.WriteLine("Type.GetTypeFromHandle(myRTHFromObject):  {0}",
            Type.GetTypeFromHandle(myRTHFromObject));

        Console.WriteLine("\nmyRTHFromObject.Equals(myRTHFromType):  {0}",
            myRTHFromObject.Equals(myRTHFromType));

        Console.WriteLine("\nmyRTHFromType.Value:  {0}",
myRTHFromType.Value);
        Console.WriteLine("myRTHFromType.GetType():  {0}",
myRTHFromType.GetType());
        Console.WriteLine("Get the type back from the handle...");
        Console.WriteLine("Type.GetTypeFromHandle(myRTHFromType):  {0}",
            Type.GetTypeFromHandle(myRTHFromType));
    }
}

/* This code example produces output similar to the following:

myRTHFromObject.Value:  799464
myRTHFromObject.GetType():  System.RuntimeTypeHandle
Get the type back from the handle...
Type.GetTypeFromHandle(myRTHFromObject):  MyClass1

myRTHFromObject.Equals(myRTHFromType):  True

myRTHFromType.Value:  799464
myRTHFromType.GetType():  System.RuntimeTypeHandle
Get the type back from the handle...
Type.GetTypeFromHandle(myRTHFromType):  MyClass1
 */
```

Properties

Expand table

Value	Gets a handle to the type represented by this instance.

Methods

Expand table

Equals(Object)	Indicates whether the specified object is equal to the current RuntimeTypeHandle structure.
Equals(RuntimeTypeHandle)	Indicates whether the specified RuntimeTypeHandle structure is equal to the current RuntimeTypeHandle structure.
FromIntPtr(IntPtr)	Returns a new RuntimeTypeHandle object created from a handle to a RuntimeType.
GetHashCode()	Returns the hash code for the current instance.
GetModuleHandle()	Gets a handle to the module that contains the type represented by the current instance.
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo with the data necessary to deserialize the type represented by the current instance.
ToIntPtr(RuntimeTypeHandle)	Returns the internal pointer representation of a RuntimeTypeHandle object.

Operators

Expand table

Equality(Object, RuntimeTypeHandle)	Indicates whether an object and a RuntimeTypeHandle structure are equal.
Equality(RuntimeTypeHandle, Object)	Indicates whether a RuntimeTypeHandle structure is equal to an object.
Inequality(Object, RuntimeTypeHandle)	Indicates whether an object and a RuntimeTypeHandle structure are not equal.
Inequality(RuntimeTypeHandle, Object)	Indicates whether a RuntimeTypeHandle structure is not equal to an object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle.Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a handle to the type represented by this instance.

C#

```
public IntPtr Value { get; }
```

Property Value

IntPtr

A handle to the type represented by this instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two RuntimeTypeHandle structures are equal.

Overloads

Expand table

Equals(Object)	Indicates whether the specified object is equal to the current RuntimeTypeHandle structure.
Equals(RuntimeTypeHandle)	Indicates whether the specified RuntimeTypeHandle structure is equal to the current RuntimeTypeHandle structure.

Equals(Object)

Indicates whether the specified object is equal to the current RuntimeTypeHandle structure.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare to the current instance.

Returns

Boolean

`true` if `obj` is a RuntimeTypeHandle structure and is equal to the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(RuntimeTypeHandle)

Indicates whether the specified RuntimeTypeHandle structure is equal to the current RuntimeTypeHandle structure.

C#

```
public bool Equals (RuntimeTypeHandle handle);
```

Parameters

`handle` RuntimeTypeHandle

The RuntimeTypeHandle structure to compare to the current instance.

Returns

Boolean

`true` if the value of `handle` is equal to the value of this instance; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle.FromIntPtr(IntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new RuntimeTypeHandle object created from a handle to a RuntimeType.

C#

```
public static RuntimeTypeHandle FromIntPtr (IntPtr value);
```

Parameters

`value` IntPtr

An IntPtr handle to a RuntimeType to create a RuntimeTypeHandle object from.

Returns

RuntimeTypeHandle

A new RuntimeTypeHandle object that corresponds to the value parameter.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeTypeHandle.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeTypeHandle.GetModuleHandle Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Gets a handle to the module that contains the type represented by the current instance.

C#

```
public ModuleHandle GetModuleHandle ();
```

Returns

ModuleHandle

A ModuleHandle structure representing a handle to the module that contains the type represented by the current instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

RuntimeTypeHandle.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo with the data necessary to deserialize the type represented by the current instance.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to be populated with serialization information.

`context` StreamingContext

(Reserved) The location where serialized data will be stored and retrieved.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

SerializationException

Value is invalid.

Remarks

The `context` parameter is reserved; it does not currently participate in this operation.

For more information about the serialization operation performed by this method, see the SerializationInfo.AddValue(String, Object, Type) method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle.ToIntPtr(RuntimeTypeHandle) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the internal pointer representation of a RuntimeTypeHandle object.

C#

```
public static IntPtr ToIntPtr (RuntimeTypeHandle value);
```

Parameters

`value` RuntimeTypeHandle

A RuntimeTypeHandle object to retrieve an internal pointer representation from.

Returns

IntPtr

An IntPtr object that represents a RuntimeTypeHandle object.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

RuntimeTypeHandle.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether an object and a RuntimeTypeHandle structure are equal.

Overloads

Expand table

Equality(Object, RuntimeTypeHandle)	Indicates whether an object and a RuntimeTypeHandle structure are equal.
Equality(RuntimeTypeHandle, Object)	Indicates whether a RuntimeTypeHandle structure is equal to an object.

Equality(Object, RuntimeTypeHandle)

Indicates whether an object and a RuntimeTypeHandle structure are equal.

C#

```
public static bool operator == (object? left, RuntimeTypeHandle right);
```

Parameters

`left` Object

An object to compare to `right`.

`right` RuntimeTypeHandle

A RuntimeTypeHandle structure to compare to `left`.

Returns

Boolean

`true` if `left` is a RuntimeTypeHandle structure and is equal to `right`; otherwise, `false`.

Remarks

Using this operator to compare two variables of type RuntimeTypeHandle causes an ambiguous overload resolution error when compiled. Use the Equals method instead.

The equivalent method for this operator is RuntimeTypeHandle.Equals(Object).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equality(RuntimeTypeHandle, Object)

Indicates whether a RuntimeTypeHandle structure is equal to an object.

C#

```
public static bool operator == (RuntimeTypeHandle left, object? right);
```

Parameters

`left` RuntimeTypeHandle

A RuntimeTypeHandle structure to compare to `right`.

`right` Object

An object to compare to `left`.

Returns

Boolean

`true` if `right` is a RuntimeTypeHandle and is equal to `left`; otherwise, `false`.

Remarks

Using this operator to compare two variables of type RuntimeTypeHandle causes an ambiguous overload resolution error when compiled. Use the Equals method instead.

The equivalent method for this operator is RuntimeTypeHandle.Equals(Object)>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

RuntimeTypeHandle.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether an object and a RuntimeTypeHandle structure are not equal.

Overloads

Expand table

Inequality(Object, RuntimeTypeHandle)	Indicates whether an object and a RuntimeTypeHandle structure are not equal.
Inequality(RuntimeTypeHandle, Object)	Indicates whether a RuntimeTypeHandle structure is not equal to an object.

Inequality(Object, RuntimeTypeHandle)

Indicates whether an object and a RuntimeTypeHandle structure are not equal.

C#

```
public static bool operator != (object? left, RuntimeTypeHandle right);
```

Parameters

`left` Object

An object to compare to `right`.

`right` RuntimeTypeHandle

A RuntimeTypeHandle structure to compare to `left`.

Returns

Boolean

`true` if `left` is a RuntimeTypeHandle and is not equal to `right`; otherwise, `false`.

Remarks

Using this operator to compare two variables of type RuntimeTypeHandle causes an ambiguous overload resolution error when compiled. Use the Equals method instead.

The equivalent method for this operator is RuntimeTypeHandle.Equals(Object).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Inequality(RuntimeTypeHandle, Object)

Indicates whether a RuntimeTypeHandle structure is not equal to an object.

C#

```
public static bool operator != (RuntimeTypeHandle left, object? right);
```

Parameters

`left` RuntimeTypeHandle

A RuntimeTypeHandle structure to compare to `right`.

`right` Object

An object to compare to `left`.

Returns

Boolean

`true` if `right` is a RuntimeTypeHandle structure and is not equal to `left`; otherwise, `false`.

Remarks

Using this operator to compare two variables of type RuntimeTypeHandle causes an ambiguous overload resolution error when compiled. Use the Equals method instead.

The equivalent method for this operator is RuntimeTypeHandle.Equals(Object).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int16**

Represents an 8-bit signed integer.

C#

```
[System.CLSCompliant(false)]
public readonly struct SByte : IComparable<sbyte>, IConvertible,
IEquatable<sbyte>, IParsable<sbyte>, ISpanParsable<sbyte>,
IUtf8SpanParsable<sbyte>,
System.Numerics.IAdditionOperators<sbyte,sbyte,sbyte>,
System.Numerics.IAdditiveIdentity<sbyte,sbyte>,
System.Numerics.IBinaryInteger<sbyte>, System.Numerics.IBinaryNumber<sbyte>,
System.Numerics.IBitwiseOperators<sbyte,sbyte,sbyte>,
System.Numerics.IComparisonOperators<sbyte,sbyte,bool>,
System.Numerics.IDecrementOperators<sbyte>,
System.Numerics.IDivisionOperators<sbyte,sbyte,sbyte>,
System.Numerics.IEqualityOperators<sbyte,sbyte,bool>,
System.Numerics.IIncrementOperators<sbyte>,
System.Numerics.IMinMaxValue<sbyte>,
System.Numerics.IModulusOperators<sbyte,sbyte,sbyte>,
System.Numerics.IMultiplicativeIdentity<sbyte,sbyte>,
System.Numerics.IMultiplyOperators<sbyte,sbyte,sbyte>,
System.Numerics.INumber<sbyte>, System.Numerics.INumberBase<sbyte>,
System.Numerics.IShiftOperators<sbyte,int,sbyte>,
System.Numerics.ISignedNumber<sbyte>,
System.Numerics.ISubtractionOperators<sbyte,sbyte,sbyte>,
System.Numerics.IUnaryNegationOperators<sbyte,sbyte>,
System.Numerics.IUnaryPlusOperators<sbyte,sbyte>
```

Inheritance Object → ValueType → SByte

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<SByte> , IConvertible , IEquatable<SByte> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<SByte> , IParsable<TSelf> , ISpanParsable<SByte> , ISpanParsable<TSelf> , IAdditionOperators<SByte,SByte,SByte> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<SByte,SByte> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<SByte> , IBinaryNumber<SByte> , IBinaryNumber<TSelf> , IBitwiseOperators<SByte,SByte,SByte> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<SByte,SByte,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<SByte> , IDecrementOperators<TSelf> , IDivisionOperators<SByte,SByte,SByte> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<SByte,SByte,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<SByte> , IIncrementOperators<TSelf> , IMinMaxValue<SByte> , IModulusOperators<SByte,SByte,SByte> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<SByte,SByte> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<SByte,SByte,SByte> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<SByte> , INumber<TSelf> , INumberBase<SByte> , INumberBase<TSelf> , IShiftOperators<SByte,Int32,SByte> , IShiftOperators<TSelf,Int32,TSelf> , ISignedNumber<SByte> , ISubtractionOperators<SByte,SByte,SByte> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<SByte,SByte> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<SByte,SByte> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<SByte> , IUtf8SpanParsable<TSelf>

Remarks

The SByte value type represents integers with values ranging from negative 128 to positive 127.

> **Important**
>
> The **SByte** type is not CLS-compliant. The CLS-compliant alternative type is **Int16**. **Byte** can be used instead to replace a positive **SByte** value that ranges from zero to **MaxValue**. For more information about CLS compliance, see **Language Independence and Language-Independent Components**.

SByte provides methods to compare instances of this type, convert the value of an instance to its string representation, and convert the string representation of a number to an instance of this type.

For information about how format specification codes control the string representation of value types, see Formatting Types.

Fields

Expand table

MaxValue	Represents the largest possible value of SByte. This field is constant.
MinValue	Represents the smallest possible value of SByte. This field is constant.

Methods

Expand table

Abs(SByte)	Computes the absolute of a value.
Clamp(SByte, SByte, SByte)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares this instance to a specified object and returns an

	indication of their relative values.
CompareTo(SByte)	Compares this instance to a specified 8-bit signed integer and returns an indication of their relative values.
CopySign(SByte, SByte)	Copies the sign of a value to the sign of another value.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(SByte, SByte)	Computes the quotient and remainder of two values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(SByte)	Returns a value indicating whether this instance is equal to a specified SByte value.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type SByte.
IsEvenInteger(SByte)	Determines if a value represents an even integral number.
IsNegative(SByte)	Determines if a value is negative.
IsOddInteger(SByte)	Determines if a value represents an odd integral number.
IsPositive(SByte)	Determines if a value is positive.
IsPow2(SByte)	Determines if a value is a power of two.
LeadingZeroCount(SByte)	Computes the number of leading zeros in a value.
Log2(SByte)	Computes the log2 of a value.
Max(SByte, SByte)	Compares two values to compute which is greater.
MaxMagnitude(SByte, SByte)	Compares two values to compute which is greater.
Min(SByte, SByte)	Compares two values to compute which is lesser.
MinMagnitude(SByte, SByte)	Compares two values to compute which is lesser.

Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.
Parse(String)	Converts the string representation of a number to its 8-bit signed integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 8-bit signed integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 8-bit signed integer equivalent.
Parse(String, NumberStyles, IFormat Provider)	Converts the string representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.
PopCount(SByte)	Computes the number of bits that are set in a value.
RotateLeft(SByte, Int32)	Rotates a value left by a given amount.
RotateRight(SByte, Int32)	Rotates a value right by a given amount.
Sign(SByte)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(SByte)	Computes the number of trailing zeros in a value.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current 8-bit signed integer instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, SByte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, SByte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, SByte)	Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit signed integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, SByte)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, SByte)	Tries to convert the span representation of a number in a specified style and culture-specific format to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, SByte)	Tries to convert the span representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, SByte)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, SByte)	Tries to convert the string representation of a number in a specified style and culture-specific format to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, SByte)	Tries to convert the string representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

Explicit Interface Implementations

Expand table

IAdditionOperators<SByte,SByte,SByte>.Addition(SByte, SByte)	Adds two values together to compute their sum.

IAdditionOperators<SByte,SByte,SByte>.CheckedAddition(SByte, SByte)	Adds two values together to compute their sum.
IAdditiveIdentity<SByte,SByte>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<SByte>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<SByte>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<SByte>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, SByte)	
IBinaryInteger<SByte>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, SByte)	
IBinaryInteger<SByte>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<SByte>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<SByte>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<SByte,SByte,SByte>.BitwiseAnd(SByte, SByte)	Computes the bitwise-and of two values.
IBitwiseOperators<SByte,SByte,SByte>.BitwiseOr(SByte, SByte)	Computes the bitwise-or of two values.
IBitwiseOperators<SByte,SByte,SByte>.ExclusiveOr(SByte, SByte)	Computes the exclusive-or of two values.
IBitwiseOperators<SByte,SByte,SByte>.OnesComplement(SByte)	Computes the ones-complement representation of a given value.
IComparisonOperators<SByte,SByte,Boolean>.GreaterThan(SByte, SByte)	Compares two values to determine which is greater.
IComparisonOperators<SByte,SByte,Boolean>.GreaterThanOrEqual(SByte, SByte)	Compares two values to determine which is greater or equal.
IComparisonOperators<SByte,SByte,Boolean>.LessThan(SByte, SByte)	Compares two values to determine which is less.
IComparisonOperators<SByte,SByte,Boolean>.LessThanOrEqual(SByte, SByte)	Compares two values to determine which is less or equal.

IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<SByte>.Checked Decrement(SByte)	Decrements a value.
IDecrementOperators<SByte>.Decrement(SByte)	Decrements a value.

IDivisionOperators<SByte,SByte,SByte>.Division(SByte, SByte)	Divides one value by another to compute their quotient.
IEqualityOperators<SByte,SByte,Boolean>.Equality(SByte, SByte)	Compares two values to determine equality.
IEqualityOperators<SByte,SByte,Boolean>.Inequality(SByte, SByte)	Compares two values to determine inequality.
IIncrementOperators<SByte>.CheckedIncrement(SByte)	Increments a value.
IIncrementOperators<SByte>.Increment(SByte)	Increments a value.
IMinMaxValue<SByte>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<SByte>.MinValue	Gets the minimum value of the current type.
IModulusOperators<SByte,SByte,SByte>.Modulus(SByte, SByte)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<SByte,SByte>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
IMultiplyOperators<SByte,SByte,SByte>.CheckedMultiply(SByte, SByte)	Multiplies two values together to compute their product.
IMultiplyOperators<SByte,SByte,SByte>.Multiply(SByte, SByte)	Multiplies two values together to compute their product.
INumber<SByte>.MaxNumber(SByte, SByte)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<SByte>.MinNumber(SByte, SByte)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<SByte>.IsCanonical(SByte)	Determines if a value is in its canonical representation.
INumberBase<SByte>.IsComplexNumber(SByte)	Determines if a value represents a complex number.
INumberBase<SByte>.IsFinite(SByte)	Determines if a value is finite.
INumberBase<SByte>.IsImaginaryNumber(SByte)	Determines if a value represents a pure imaginary number.
INumberBase<SByte>.IsInfinity(SByte)	Determines if a value is infinite.

INumberBase<SByte>.IsInteger(SByte)	Determines if a value represents an integral number.
INumberBase<SByte>.IsNaN(SByte)	Determines if a value is NaN.
INumberBase<SByte>.IsNegativeInfinity(SByte)	Determines if a value is negative infinity.
INumberBase<SByte>.IsNormal(SByte)	Determines if a value is normal.
INumberBase<SByte>.IsPositiveInfinity(SByte)	Determines if a value is positive infinity.
INumberBase<SByte>.IsRealNumber(SByte)	Determines if a value represents a real number.
INumberBase<SByte>.IsSubnormal(SByte)	Determines if a value is subnormal.
INumberBase<SByte>.IsZero(SByte)	Determines if a value is zero.
INumberBase<SByte>.MaxMagnitudeNumber(SByte, SByte)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<SByte>.MinMagnitudeNumber(SByte, SByte)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<SByte>.One	Gets the value `1` for the type.
INumberBase<SByte>.Radix	Gets the radix, or base, for the type.
INumberBase<SByte>.TryConvertFromChecked<TOther>(TOther, SByte)	
INumberBase<SByte>.TryConvertFromSaturating<TOther>(TOther, SByte)	
INumberBase<SByte>.TryConvertFromTruncating<TOther>(TOther, SByte)	
INumberBase<SByte>.TryConvertToChecked<TOther>(SByte, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<SByte>.TryConvertToSaturating<TOther>(SByte, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

INumberBase<SByte>.TryConvertToTruncating<TOther>(SByte, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<SByte>.Zero	Gets the value `0` for the type.
IShiftOperators<SByte,Int32,SByte>.LeftShift(SByte, Int32)	Shifts a value left by a given amount.
IShiftOperators<SByte,Int32,SByte>.RightShift(SByte, Int32)	Shifts a value right by a given amount.
IShiftOperators<SByte,Int32,SByte>.UnsignedRightShift(SByte, Int32)	Shifts a value right by a given amount.
ISignedNumber<SByte>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<SByte,SByte,SByte>.CheckedSubtraction(SByte, SByte)	Subtracts two values to compute their difference.
ISubtractionOperators<SByte,SByte,SByte>.Subtraction(SByte, SByte)	Subtracts two values to compute their difference.
IUnaryNegationOperators<SByte,SByte>.CheckedUnaryNegation(SByte)	Computes the checked unary negation of a value.
IUnaryNegationOperators<SByte,SByte>.UnaryNegation(SByte)	Computes the unary negation of a value.
IUnaryPlusOperators<SByte,SByte>.UnaryPlus(SByte)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Byte
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of SByte. This field is constant.

C#

```
public const sbyte MaxValue = 127;
```

Field Value

***Value** = 127*

SByte

Examples

The following example uses the MinValue and MaxValue fields to verify that an Int64 value is within the range of the SByte type before it performs a type conversion. This verification prevents an OverflowException at run time.

C#

```
long longValue = -130;
sbyte byteValue;

if (longValue <= sbyte.MaxValue &&
    longValue >= sbyte.MinValue)
{
   byteValue = (sbyte) longValue;
   Console.WriteLine("Converted long integer value to {0}.", byteValue);
}
else
{
   sbyte rangeLimit;
   string relationship;

   if (longValue > sbyte.MaxValue)
   {
      rangeLimit = sbyte.MaxValue;
```

```
        relationship = "greater";
    }
    else
    {
        rangeLimit = sbyte.MinValue;
        relationship = "less";
    }

    Console.WriteLine("Conversion failure: {0:n0} is {1} than {2}.",
                      longValue,
                      relationship,
                      rangeLimit);
}
```

Remarks

The value of this constant is 127; that is, hexadecimal 0x7F.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of SByte. This field is constant.

C#

```
public const sbyte MinValue = -128;
```

Field Value

***Value* = -128**

SByte

Examples

The following example uses the MinValue and MaxValue fields to verify that an Int64 value is within the range of the SByte type before it performs a type conversion. This verification prevents an OverflowException at run time.

C#

```
long longValue = -130;
sbyte byteValue;

if (longValue <= sbyte.MaxValue &&
    longValue >= sbyte.MinValue)
{
   byteValue = (sbyte) longValue;
   Console.WriteLine("Converted long integer value to {0}.", byteValue);
}
else
{
   sbyte rangeLimit;
   string relationship;

   if (longValue > sbyte.MaxValue)
   {
      rangeLimit = sbyte.MaxValue;
```

```
        relationship = "greater";
    }
    else
    {
        rangeLimit = sbyte.MinValue;
        relationship = "less";
    }

    Console.WriteLine("Conversion failure: {0:n0} is {1} than {2}.",
                      longValue,
                      relationship,
                      rangeLimit);
}
```

Remarks

The value of this constant is -128; that is, hexadecimal 0x80.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Abs(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static sbyte Abs (sbyte value);
```

Parameters

`value` SByte

The value for which to get its absolute.

Returns

SByte

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Clamp(SByte, SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static sbyte Clamp (sbyte value, sbyte min, sbyte max);
```

Parameters

`value` SByte

The value to clamp.

`min` SByte

The inclusive minimum to which `value` should clamp.

`max` SByte

The inclusive maximum to which `value` should clamp.

Returns

SByte

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or SByte and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CompareTo(SByte)	Compares this instance to a specified 8-bit signed integer and returns an indication of their relative values.

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? obj);
```

Parameters

obj Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `obj`.

Expand table

Return Value	Description
Less than zero	This instance is less than `obj`.
Zero	This instance is equal to `obj`.
Greater than zero	This instance is greater than `obj`, or `obj` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not an SByte.

Remarks

Any instance of SByte, regardless of its value, is considered greater than `null`.

The `obj` parameter must be `null` or an instance of SByte; otherwise, an exception is thrown.

See also

- Byte
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

CompareTo(SByte)

Compares this instance to a specified 8-bit signed integer and returns an indication of their relative values.

```
C#
public int CompareTo (sbyte value);
```

Parameters

`value` SByte

An 8-bit signed integer to compare.

Returns

Int32

A signed integer that indicates the relative order of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                      "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
```

```
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;
```

```
    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the SByte.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine if its compiler performs implicit widening conversions on numeric types.

See also

- IComparable
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.CopySign(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

```
C#
public static sbyte CopySign (sbyte value, sbyte sign);
```

Parameters

`value` SByte

The value whose magnitude is used in the result.

`sign` SByte

The value whose sign is used in the result.

Returns

SByte

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static sbyte CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

SByte

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.CreateSaturating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static sbyte CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

TOther

The type of `value`.

Parameters

value TOther

The value that's used to create the instance of `TSelf`.

Returns

SByte

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.CreateTruncating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static sbyte CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

SByte

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.DivRem(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (sbyte Quotient, sbyte Remainder) DivRem (sbyte left, sbyte
right);
```

Parameters

`left` SByte

The value which `right` divides.

`right` SByte

The value which divides `left`.

Returns

ValueTuple<SByte,SByte>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or SByte.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(SByte)	Returns a value indicating whether this instance is equal to a specified SByte value.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of SByte and equals the value of this instance; otherwise, `false`.

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(SByte)

Returns a value indicating whether this instance is equal to a specified SByte value.

C#

```
public bool Equals (sbyte obj);
```

Parameters

`obj` SByte

An SByte value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

SByte.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type SByte.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, SByte.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IsEvenInteger(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (sbyte value);
```

Parameters

value SByte

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IsNegative(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
public static bool IsNegative (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Remarks

A return value of `false` does not imply that IsPositive(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IsOddInteger(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (sbyte value);
```

Parameters

value SByte

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IsPositive(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

C#

```
public static bool IsPositive (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IsPow2(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.LeadingZeroCount(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static sbyte LeadingZeroCount (sbyte value);
```

Parameters

`value` SByte

The value whose leading zeroes are to be counted.

Returns

SByte

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Log2(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static sbyte Log2 (sbyte value);
```

Parameters

`value` SByte

The value whose log2 is to be computed.

Returns

SByte

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Max(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static sbyte Max (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.MaxMagnitude(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static sbyte MaxMagnitude (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Min(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static sbyte Min (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.MinMagnitude(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static sbyte MinMagnitude (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 8-bit signed integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 8-bit signed integer equivalent.
Parse(String)	Converts the string representation of a number to its 8-bit signed integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 8-bit signed integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> | | |
> |---|---|
> | CLS-compliant alternative | System.Int16.Parse(String, NumberStyles, IFormatProvider) |

Converts the string representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.

```
C#

public static sbyte Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains the number to convert. The string is interpreted by using the style specified by `style`.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. If `provider` is `null`, the thread current culture is used.

Returns

SByte

An 8-bit signed byte value that is equivalent to the number specified in the `s` parameter.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber.

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in a format that is compliant with `style`.

OverflowException

`s` represents a number that is less than SByte.MinValue or greater than SByte.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example illustrates the use of the Parse(String, NumberStyles, IFormatProvider) method to convert various string representations of numbers to signed integer values.

C#

```
using System;
using System.Globalization;

public class SByteConversion
{
   NumberFormatInfo provider = NumberFormatInfo.CurrentInfo;

   public static void Main()
   {
      string stringValue;
      NumberStyles style;

      stringValue = "   123   ";
      style = NumberStyles.None;
      CallParseOperation(stringValue, style);
```

```
        stringValue = "000,000,123";
        style = NumberStyles.Integer | NumberStyles.AllowThousands;
        CallParseOperation(stringValue, style);

        stringValue = "-100";
        style = NumberStyles.AllowLeadingSign;
        CallParseOperation(stringValue, style);

        stringValue = "100-";
        style = NumberStyles.AllowLeadingSign;
        CallParseOperation(stringValue, style);

        stringValue = "100-";
        style = NumberStyles.AllowTrailingSign;
        CallParseOperation(stringValue, style);

        stringValue = "$100";
        style = NumberStyles.AllowCurrencySymbol;
        CallParseOperation(stringValue, style);

        style = NumberStyles.Integer;
        CallParseOperation(stringValue, style);

        style = NumberStyles.AllowDecimalPoint;
        CallParseOperation("100.0", style);

        stringValue = "1e02";
        style = NumberStyles.AllowExponent;
        CallParseOperation(stringValue, style);

        stringValue = "(100)";
        style = NumberStyles.AllowParentheses;
        CallParseOperation(stringValue, style);
    }

    private static void CallParseOperation(string stringValue,
                                           NumberStyles style)
    {
        sbyte number;

        if (stringValue == null)
            Console.WriteLine("Cannot parse a null string...");

        try
        {
            number = sbyte.Parse(stringValue, style);
            Console.WriteLine("SByte.Parse('{0}', {1})) = {2}",
                              stringValue, style, number);
        }
        catch (FormatException)
        {
            Console.WriteLine("'{0}' and {1} throw a FormatException",
                              stringValue, style);
        }
        catch (OverflowException)
```

```
        {
            Console.WriteLine("'{0}' is outside the range of a signed byte",
                              stringValue);
        }
    }
}
// The example displays the following information to the console:
//       '   123   ' and None throw a FormatException
//       SByte.Parse('000,000,123', Integer, AllowThousands)) = 123
//       SByte.Parse('-100', AllowLeadingSign)) = -100
//       '100-' and AllowLeadingSign throw a FormatException
//       SByte.Parse('100-', AllowTrailingSign)) = -100
//       SByte.Parse('$100', AllowCurrencySymbol)) = 100
//       '$100' and Integer throw a FormatException
//       SByte.Parse('100.0', AllowDecimalPoint)) = 100
//       SByte.Parse('1e02', AllowExponent)) = 100
//       SByte.Parse('(100)', AllowParentheses)) = -100
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive or negative sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

If `style` includes AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.

Element	Description
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0-9 if `style` includes the NumberStyles.AllowExponent flag, or one or more occurrences of the digit 0 if it does not. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

Note

Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, if style does not include the NumberStyles.AllowExponent flag, *fractional_digits* must consist of only one or more 0 digits; otherwise, an OverflowException is thrown.
NumberStyles.AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign before *digits*.
NumberStyles.AllowTrailingSign	A positive sign after *digits*.
NumberStyles.AllowParentheses	Parentheses before and after *digits* to indicate a negative value.
NumberStyles.AllowThousands	The group separator (,) element. Although the group separator can appear in `s`, it must be preceded by only one or more 0 digits.
NumberStyles.AllowCurrencySymbol	The currency ($) element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal digits are 0-9, a-f, and A-F. The only other flags that can be combined with it are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If the `s` parameter is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the parse operation to throw an exception.

If `s` represents a hexadecimal number, the Parse(String, NumberStyles) method interprets the high-order bit of the byte as a sign bit.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the span representation of a number that is in a specified style and culture-specific format to its 8-bit signed equivalent.

C#

```
public static sbyte Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert. The span is interpreted by using the style specified by `style`.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. If `provider` is `null`, the thread current culture is used.

Returns

SByte

An 8-bit signed byte value that is equivalent to the number specified in the `s` parameter.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static sbyte Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

SByte

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int16.Parse(String)**

Converts the string representation of a number in a specified culture-specific format to its 8-bit signed integer equivalent.

```
C#
public static sbyte Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents a number to convert. The string is interpreted using the Integer style.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`. If `provider` is `null`, the thread current culture is used.

Returns

SByte

An 8-bit signed integer that is equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example defines a custom NumberFormatInfo object that defines the tilde (~) as the negative sign. It then parses a number of numeric strings using this custom NumberFormatInfo object as well as a CultureInfo object that represents the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      NumberFormatInfo nf = new NumberFormatInfo();
      nf.NegativeSign = "~";

      string[] values = { "-103", "+12", "~16", "  1", "~255" };
      IFormatProvider[] providers = { nf, CultureInfo.InvariantCulture };

      foreach (IFormatProvider provider in providers)
      {
         Console.WriteLine("Conversions using {0}:", ((object)
provider).GetType().Name);
         foreach (string value in values)
         {
            try {
               Console.WriteLine("   Converted '{0}' to {1}.",
                                 value, SByte.Parse(value, provider));
            }
            catch (FormatException) {
               Console.WriteLine("   Unable to parse '{0}'.", value);
            }
            catch (OverflowException) {
               Console.WriteLine("   '{0}' is out of range of the SByte
type.", value);
            }
         }
```

```
        }
    }
}
// The example displays the following output:
//       Conversions using NumberFormatInfo:
//          Unable to parse '-103'.
//          Converted '+12' to 12.
//          Converted '~16' to -16.
//          Converted '  1' to 1.
//          '~255' is out of range of the SByte type.
//       Conversions using CultureInfo:
//          Converted '-103' to -103.
//          Converted '+12' to 12.
//          Unable to parse '~16'.
//          Converted '  1' to 1.
//          Unable to parse '~255'.
```

Remarks

The `s` parameter contains a number of the form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces with a leading sign are allowed. To explicitly define the style elements with the culture-specific formatting information that can be present in `s`, use the Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int16.Parse(String)**

Converts the string representation of a number to its 8-bit signed integer equivalent.

```C#
public static sbyte Parse (string s);
```

Parameters

s String

A string that represents a number to convert. The string is interpreted using the Integer style.

Returns

SByte

An 8-bit signed integer that is equivalent to the number contained in the `s` parameter.

Exceptions

ArgumentException

`s` is `null`.

FormatException

`s` does not consist of an optional sign followed by a sequence of digits (zero through nine).

OverflowException

`s` represents a number less than SByte.MinValue or greater than SByte.MaxValue.

Examples

The following example demonstrates how to convert a string value into a signed byte value using the Parse method. The resulting signed byte value is then displayed to the console.

```C#
// Define an array of numeric strings.
string[] values = { "-16", "  -3", "+ 12", " +12 ", "  12  ",
                    "+120", "(103)", "192", "-160" };

// Parse each string and display the result.
foreach (string value in values)
```

```
{
   try {
      Console.WriteLine("Converted '{0}' to the SByte value {1}.",
                        value, SByte.Parse(value));
   }
   catch (FormatException) {
      Console.WriteLine("'{0}' cannot be parsed successfully by SByte
type.",
                        value);
   }
   catch (OverflowException) {
      Console.WriteLine("'{0}' is out of range of the SByte type.",
                        value);
   }
}
// The example displays the following output:
//       Converted '-16' to the SByte value -16.
//       Converted '  -3' to the SByte value -3.
//       '+ 12' cannot be parsed successfully by SByte type.
//       Converted ' +12 ' to the SByte value 12.
//       Converted '  12  ' to the SByte value 12.
//       Converted '+120' to the SByte value 120.
//       '(103)' cannot be parsed successfully by SByte type.
//       '192' is out of range of the SByte type.
//       '-160' is out of range of the SByte type.
```

Remarks

The `s` parameter contains a number of the form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign.
digits	A sequence of digits ranging from 0 to 9.

The `s` parameter is interpreted using the NumberStyles.Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces with a leading positive or negative sign are allowed. To explicitly define the style elements that can

be present in `s`, use either the Parse(String, NumberStyles) or the Parse(String, NumberStyles, IFormatProvider) method.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string by using the formatting information of some other culture, use the Parse(String, NumberStyles, IFormatProvider) method.

See also

- Parsing Numeric Strings in .NET
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static sbyte Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

SByte

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static sbyte Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

utf8Text ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

SByte

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int16.Parse(String)**

Converts the string representation of a number in a specified style to its 8-bit signed integer equivalent.

C#

```
public static sbyte Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string that contains a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

Returns

SByte

An 8-bit signed integer that is equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in a format that is compliant with `style`.

OverflowException

`s` represents a number less than SByte.MinValue or greater than SByte.MaxValue.

-or-

`s` includes non-zero, fractional digits.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example parses string representations of SByte values with the Parse(String, NumberStyles) method. The current culture for the example is en-US.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      NumberStyles style;
      sbyte number;

      // Parse value with no styles allowed.
```

```
      string[] values1 = { " 121 ", "121", "-121" };
      style = NumberStyles.None;
      Console.WriteLine("Styles: {0}", style.ToString());
      foreach (string value in values1)
      {
         try {
            number = SByte.Parse(value, style);
            Console.WriteLine("   Converted '{0}' to {1}.", value,
number);
         }
         catch (FormatException) {
            Console.WriteLine("   Unable to parse '{0}'.", value);
         }
      }
      Console.WriteLine();

      // Parse value with trailing sign.
      style = NumberStyles.Integer | NumberStyles.AllowTrailingSign;
      string[] values2 = { " 103+", " 103 +", "+103", "(103)", "   +103
" };
      Console.WriteLine("Styles: {0}", style.ToString());
      foreach (string value in values2)
      {
         try {
            number = SByte.Parse(value, style);
            Console.WriteLine("   Converted '{0}' to {1}.", value,
number);
         }
         catch (FormatException) {
            Console.WriteLine("   Unable to parse '{0}'.", value);
         }
         catch (OverflowException) {
            Console.WriteLine("   '{0}' is out of range of the SByte
type.", value);
         }
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//       Styles: None
//          Unable to parse ' 121 '.
//          Converted '121' to 121.
//          Unable to parse '-121'.
//
//       Styles: Integer, AllowTrailingSign
//          Converted ' 103+' to 103.
//          Converted ' 103 +' to 103.
//          Converted '+103' to 103.
//          Unable to parse '(103)'.
//          Converted '   +103   ' to 103.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive or negative sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may contain the following elements:

[*ws*]*hexdigits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if style includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyPositivePattern property of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0-9 if `style` includes the NumberStyles.AllowExponent flag, or one or more occurrences of the digit 0 if it does not. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.

Element	Description
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	One or more occurrences of the digit 0-9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite NumberStyles values	Elements permitted in *s* in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, if `style` does not include the NumberStyles.AllowExponent flag, *fractional_digits* must consist of only one or more 0 digits; otherwise, an OverflowException is thrown.
NumberStyles.AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A positive sign before *digits*.

Non-composite NumberStyles values	Elements permitted in *s* in addition to digits
NumberStyles.AllowTrailingSign	A positive sign after *digits*.
NumberStyles.AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
NumberStyles.AllowThousands	The group separator (,) element. Although the group separator can appear in `s`, it must be preceded by only one or more 0 digits.
NumberStyles.AllowCurrencySymbol	The currency ($) element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal digits are 0-9, a-f, and A-F. A prefix such as "0x" is not supported and causes the parse operation to fail. The only other flags that can be combined included in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If the `s` parameter is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the parse operation to throw an exception.

If `s` represents a hexadecimal number, the Parse(String, NumberStyles) method interprets the high-order bit of the byte as a sign bit.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To use the formatting information of some other culture, call the Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.PopCount(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static sbyte PopCount (sbyte value);
```

Parameters

`value` SByte

The value whose set bits are to be counted.

Returns

SByte

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.RotateLeft(SByte, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

C#

```
public static sbyte RotateLeft (sbyte value, int rotateAmount);
```

Parameters

`value` SByte

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

SByte

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.RotateRight(SByte, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static sbyte RotateRight (sbyte value, int rotateAmount);
```

Parameters

`value` SByte

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

SByte

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.Sign(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (sbyte value);
```

Parameters

`value` SByte

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a negative sign if the value is negative, and a sequence of digits ranging from 0 to 9 with no leading zeroes.

Examples

The following example displays an SByte value using the default ToString() method. It also displays the string representations of the SByte value that results from using a number of standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      sbyte value = -123;
      // Display value using default ToString method.
      Console.WriteLine(value.ToString());            // Displays -123
      // Display value using some standard format specifiers.
      Console.WriteLine(value.ToString("G"));         // Displays -123
      Console.WriteLine(value.ToString("C"));         // Displays
($-123.00)
      Console.WriteLine(value.ToString("D"));         // Displays -123
      Console.WriteLine(value.ToString("F"));         // Displays -123.00
      Console.WriteLine(value.ToString("N"));         // Displays -123.00
      Console.WriteLine(value.ToString("X"));         // Displays 85
   }
}
```

Remarks

The ToString() method formats an SByte value in the default ("G", or general) format of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

To use format	For culture	Use the overload
A specific format	A specific culture	ToString(String, IFormatProvider)

The return value is formatted using the general numeric format specifier ("G") The string representation of the SByte value includes a negative sign if its value is negative, and a sequence of digits ranging from 0 to 9 without leading zeros. The negative sign is defined by the NumberFormatInfo object for the current culture.

To define the formatting of the signed byte value's string representation, call the ToString(String) method.

See also

- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance, as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example defines a custom NumberFormatInfo object and assigns the "~" character to its NegativeSign property. The example then uses this custom object as well as the NumberFormatInfo object of the invariant culture to format a series of SByte values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define a custom NumberFormatInfo object with "~" as its negative
sign.
      NumberFormatInfo nfi = new NumberFormatInfo();
      nfi.NegativeSign = "~";

      // Initialize an array of SByte values.
      sbyte[] bytes = { -122, 17, 124 };

      // Display the formatted result using the custom provider.
      Console.WriteLine("Using the custom NumberFormatInfo object:");
      foreach (sbyte value in bytes)
         Console.WriteLine(value.ToString(nfi));

      Console.WriteLine();

      // Display the formatted result using the invariant culture.
      Console.WriteLine("Using the invariant culture:");
      foreach (sbyte value in bytes)
```

```
Console.WriteLine(value.ToString(NumberFormatInfo.InvariantInfo));
    }
}
// The example displays the following output:
//       Using the custom NumberFormatInfo object:
//       ~122
//       17
//       124
//
//       Using the invariant culture:
//       -122
//       17
//       124
```

Remarks

The ToString(IFormatProvider) method formats an SByte value in the default ("G", or general) format of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string returned by this method. If `provider` is `null`, the SByte value is formatted using the NumberFormatInfo object of the current culture. The only property of the NumberFormatInfo object that controls the string representation of the SByte value using the general format specifier is NumberFormatInfo.NegativeSign, which defines the character that represents the negative sign.

The `provider` parameter can be one of the following:

- A CultureInfo object that represents the culture that supplies formatting information.
- The NumberFormatInfo object that supplies formatting information.

- A custom object that implements IFormatProvider. Its GetFormat method returns the NumberFormatInfo object that supplies formatting information.

See also

- Formatting Types in .NET
- Standard Numeric Format Strings
- Custom Numeric Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid.

Examples

The following example initializes an array of SByte values and displays them by using each standard format string and some custom format strings.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      sbyte[] values = { -124, 0, 118 };
      string[] specifiers = { "G", "C", "D3", "E2", "e3", "F",
                              "N", "P", "X", "00.0", "#.0",
                              "000;(0);**Zero**" };

      foreach (sbyte value in values)
      {
         foreach (string specifier in specifiers)
            Console.WriteLine("{0}: {1}", specifier,
value.ToString(specifier));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       G: -124
//       C: ($124.00)
//       D3: -124
//       E2: -1.24E+002
//       e3: -1.240e+002
//       F: -124.00
//       N: -124.00
//       P: -12,400.00 %
//       X: 84
//       00.0: -124.0
//       #.0: -124.0
//       000;(0);**Zero**: (124)
//
//       G: 0
//       C: $0.00
```

```
//        D3: 000
//        E2: 0.00E+000
//        e3: 0.000e+000
//        F: 0.00
//        N: 0.00
//        P: 0.00 %
//        X: 0
//        00.0: 00.0
//        #.0: .0
//        000;(0);**Zero**: **Zero**
//
//        G: 118
//        C: $118.00
//        D3: 118
//        E2: 1.18E+002
//        e3: 1.180e+002
//        F: 118.00
//        N: 118.00
//        P: 11,800.00 %
//        X: 76
//        00.0: 118.0
//        #.0: 118.0
//        000;(0);**Zero**: 118
```

Remarks

The ToString(String) method formats an SByte value in a specified format by using the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid standard numeric format specifier, or any combination of custom numeric format specifiers. If `format` is equal to String.Empty or is `null`, the return value of the current SByte object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The format of the returned string is determined by the NumberFormatInfo object for the current culture. Depending on the `format` parameter, this object controls symbols such as the negative sign, the group separator, and the decimal point symbol in the output string. To provide formatting information for cultures other than the current culture, call the ToString(String, IFormatProvider) overload.

See also

- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Standard Numeric Format Strings
- Custom Numeric Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

```C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A standard or custom numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid.

Examples

The following example displays both a positive and a negative SByte value using the standard numeric format specifiers and a number of specific CultureInfo objects.

```C#
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define cultures whose formatting conventions are to be used.
      CultureInfo[] cultures = { CultureInfo.CreateSpecificCulture("en-
```

```
US"),
                                    CultureInfo.CreateSpecificCulture("fr-
FR"),
                                    CultureInfo.CreateSpecificCulture("es-
ES") };
      sbyte positiveNumber = 119;
      sbyte negativeNumber = -45;
      string[] specifiers = {"G", "C", "D4", "E2", "F", "N", "P", "X2"};

      foreach (string specifier in specifiers)
      {
         foreach (CultureInfo culture in cultures)
            Console.WriteLine("{0,2} format using {1} culture: {2, 16}
{3, 16}",
                              specifier, culture.Name,
                              positiveNumber.ToString(specifier,
culture),
                              negativeNumber.ToString(specifier,
culture));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//     G format using en-US culture:              119              -45
//     G format using fr-FR culture:              119              -45
//     G format using es-ES culture:              119              -45
//
//     C format using en-US culture:          $119.00         ($45.00)
//     C format using fr-FR culture:         119,00 €         -45,00 €
//     C format using es-ES culture:         119,00 €         -45,00 €
//
//    D4 format using en-US culture:             0119            -0045
//    D4 format using fr-FR culture:             0119            -0045
//    D4 format using es-ES culture:             0119            -0045
//
//    E2 format using en-US culture:        1.19E+002       -4.50E+001
//    E2 format using fr-FR culture:        1,19E+002       -4,50E+001
//    E2 format using es-ES culture:        1,19E+002       -4,50E+001
//
//     F format using en-US culture:           119.00           -45.00
//     F format using fr-FR culture:           119,00           -45,00
//     F format using es-ES culture:           119,00           -45,00
//
//     N format using en-US culture:           119.00           -45.00
//     N format using fr-FR culture:           119,00           -45,00
//     N format using es-ES culture:           119,00           -45,00
//
//     P format using en-US culture:      11,900.00 %      -4,500.00 %
//     P format using fr-FR culture:      11 900,00 %      -4 500,00 %
//     P format using es-ES culture:      11.900,00 %      -4.500,00 %
//
//    X2 format using en-US culture:               77               D3
//    X2 format using fr-FR culture:               77               D3
//    X2 format using es-ES culture:               77               D3
```

Remarks

The ToString(String, IFormatProvider) method formats an SByte value in a specified format of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be any valid standard numeric format specifier, or any combination of custom numeric format specifiers. If `format` is equal to String.Empty or is `null`, the return value of the current SByte object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string returned by this method. When the ToString(String, IFormatProvider) method is invoked, it calls the `provider` parameter's IFormatProvider.GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information for formatting the `value` parameter, such as the negative sign symbol, the group separator symbol, or the decimal point symbol. There are three ways to use the `provider` parameter to supply formatting information to the ToString(String, IFormatProvider) method:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the formatting of the returned string is based on the NumberFormatInfo object of the current culture.

See also

- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.TrailingZeroCount(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static sbyte TrailingZeroCount (sbyte value);
```

Parameters

`value` SByte

The value whose trailing zeroes are to be counted.

Returns

SByte

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current 8-bit signed integer instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current 8-bit signed integer instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 8-bit signed integer equivalent. A return code indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, SByte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, SByte)	Tries to convert the span representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, SByte)	Tries to convert the string representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, SByte)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, SByte)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, SByte)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, SByte)	Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit signed integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, SByte)	Tries to convert the span representation of a number in a specified style and culture-specific format to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.
TryParse(String, NumberStyles, IFormatProvider, SByte)	Tries to convert the string representation of a number in a specified style and culture-specific format to its

	SByte equivalent, and returns a value that indicates whether the conversion succeeded.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, SByte)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out sbyte result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` SByte

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, SByte)

> Important
>
> This API is not CLS-compliant.

Tries to convert the span representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out sbyte result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`result` SByte

When this method returns, contains the 8-bit signed integer value that is equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in the correct format, or represents a number that is less than SByte.MinValue or greater than SByte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, SByte)

> Important
>
> This API is not CLS-compliant.

Tries to convert the string representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#
public static bool TryParse (string? s, out sbyte result);
```

Parameters

`s` String

A string that contains a number to convert.

`result` SByte

When this method returns, contains the 8-bit signed integer value that is equivalent to the number contained in `s` if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in the correct format, or represents a number that is less than SByte.MinValue or greater than SByte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example tries to convert the strings in an array to SByte values by calling the TryParse(String, SByte) method.

```
C#
string[] numericStrings = {"-3.6", "12.8", "+16.7", "    3    ", "(17)",
                           "-17", "+12", "18-", "987", "1,024", "  127
```

```
"};
sbyte number;
foreach (string numericString in numericStrings)
{
   if (sbyte.TryParse(numericString, out number))
      Console.WriteLine("Converted '{0}' to {1}.", numericString,
number);
   else
      Console.WriteLine("Cannot convert '{0}' to an SByte.",
numericString);
}
// The example displays the following output to the console:
//       Cannot convert '-3.6' to an SByte.
//       Cannot convert '12.8' to an SByte.
//       Cannot convert '+16.7' to an SByte.
//       Converted '    3   ' to 3.
//       Cannot convert '(17)' to an SByte.
//       Converted '-17' to -17.
//       Converted '+12' to 12.
//       Cannot convert '18-' to an SByte.
//       Cannot convert '987' to an SByte.
//       Cannot convert '1,024' to an SByte.
//       Converted '  127 ' to 127.
```

Remarks

The SByte.TryParse(String, SByte) method is like the SByte.Parse(String) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `value` is invalid and cannot be successfully parsed.

The `s` parameter should be the string representation of a decimal number in the following form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture.

Element	Description
digits	A sequence of decimal digits ranging from 0 to 9.

> **Note**
>
> The string specified by the `value` parameter cannot contain any group separators or decimal separator, and it cannot have a decimal portion.

The `s` parameter is interpreted by using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces with a leading sign are allowed. To explicitly define the style elements with the culture-specific formatting information that can be present in `value`, call the TryParse(String, NumberStyles, IFormatProvider, SByte) method.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object for the current culture. For more information, see NumberFormatInfo.CurrentInfo.

This overload interprets all digits in the `value` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the TryParse(String, NumberStyles, IFormatProvider, SByte) overload instead.

See also

- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, SByte)

Tries to parse a span of characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out sbyte result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` SByte

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, SByte)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
sbyte result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` SByte

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, SByte)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
sbyte result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` SByte

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, SByte)

Tries to convert a UTF-8 character span containing the string representation of a number to its 8-bit signed integer equivalent.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text, out sbyte
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` SByte

When this method returns, contains the 8-bit signed integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, SByte)

> Important
>
> This API is not CLS-compliant.

Tries to convert the span representation of a number in a specified style and culture-specific format to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
sbyte result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` SByte

When this method returns, contains the 8-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than SByte.MinValue or greater than SByte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, SByte)

Important

This API is not CLS-compliant.

CLS-compliant alternative **System.Int16.TryParse(String, Int16)**

Tries to convert the string representation of a number in a specified style and culture-specific format to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out sbyte result);
```

Parameters

`s` String

A string representing a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` SByte

When this method returns, contains the 8-bit signed integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than SByte.MinValue or greater than SByte.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, SByte) method with a number of different string and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "106";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-106";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);

      numericString = "103.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "103.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "12E-01";
      CallTryParse(numericString, styles);

      numericString = "12E01";
      CallTryParse(numericString, styles);

      numericString = "C8";
```

```
        CallTryParse(numericString, NumberStyles.HexNumber);

        numericString = "0x8C";
        CallTryParse(numericString, NumberStyles.HexNumber);
    }

    private static void CallTryParse(string stringToConvert, NumberStyles
styles)
    {
        sbyte number;
        bool result = SByte.TryParse(stringToConvert, styles,
                                     CultureInfo.InvariantCulture, out
number);
        if (result)
            Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
        else
            Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
    }
}
// The example displays the following output:
//       Converted '106' to 106.
//       Attempted conversion of '-106' failed.
//       Converted '103.00' to 103.
//       Attempted conversion of '103.72' failed.
//       Converted '10E-01' to 1.
//       Attempted conversion of '12E-01' failed.
//       Converted '12E01' to 120.
//       Converted 'C8' to -56.
//       Attempted conversion of '0x8C' failed.
```

Remarks

The TryParse(String, NumberStyles, IFormatProvider, SByte) method is like the Parse(String, NumberStyles, IFormatProvider) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `value` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `value` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `value` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

If the `style` parameter includes AllowHexSpecifier, the `value` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `value` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `value` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `value` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the start of `value` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `value` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `value` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
,	A culture-specific group separator. The group separator of the culture specified by `provider` can appear in `value` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `value` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `value` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `value` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `value` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.

Element	Description
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `value`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
None	Decimal digits only.
AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits, or the method returns `false`.
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*. If `value` represents a number in exponential notation, it cannot have a non-zero, fractional component.
AllowLeadingWhite	The *ws* element at the start of `value`.
AllowTrailingWhite	The *ws* element at the end of `value`.
AllowLeadingSign	The *sign* element before *digits*.
AllowTrailingSign	The *sign* element after *digits*.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
Currency	All elements. However, `value` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `value`, *sign* at the start of `value`, and the decimal point (.) symbol. The `value` parameter can also use exponential notation.
Number	The *ws*, *sign*, group separator (,), and decimal point (.) elements.
Any	All elements. However, `value` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `value` must be a hexadecimal value. Valid hexadecimal digits are 0-9, a-f, and A-F. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, HexNumber, that includes both white-space flags.)

> **Note**
>
> If `value` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `value`. The `provider` parameter can be any one of the following:

- A CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- A NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- A custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String, NumberStyles, IFormatProvider)
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is unused.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is unused.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to do so throws an InvalidCastException.

C#

```
DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

None. This conversion is not supported.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is unused.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

```
C#
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, unchanged.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

`type` Type

The Type to which to convert this SByte value.

`provider` IFormatProvider

A IFormatProvider implementation that provides information about the format of the returned value.

Returns

Object

The value of the current instance, converted to an object of type `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToUInt16(IFormatProvider).

```
C#
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToUInt16(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToUInt32(IFormatProvider).

```
C#
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToUInt32(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToUInt64(IFormatProvider).

```
C#
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the SByte instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToUInt64(SByte) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.IAdditionOperators<SByte,SByte,SByte>.AdditionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static sbyte IAdditionOperators<sbyte,sbyte,sbyte>.operator + (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to which `right` is added.

`right` SByte

The value that's added to `left`.

Returns

SByte

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IAdditionOperators<SByte,SByte,SByte>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static sbyte IAdditionOperators<sbyte,sbyte,sbyte>.op_CheckedAddition (sbyte
left, sbyte right);
```

Parameters

`left` SByte

The value to which `right` is added.

`right` SByte

The value that's added to `left`.

Returns

SByte

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IAdditiveIdentity<SByte,SByte>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static sbyte
System.Numerics.IAdditiveIdentity<System.SByte,System.SByte>.AdditiveIdentit
y { get; }
```

Property Value

SByte

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.IBinaryInteger<SByte>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

```
C#

int IBinaryInteger<sbyte>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.IBinaryInteger<SByte>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<sbyte>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

SByte.IBinaryInteger<SByte>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<sbyte>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out sbyte value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` SByte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

SByte.IBinaryInteger<SByte>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<sbyte>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out sbyte value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` SByte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

SByte.IBinaryInteger<SByte>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<sbyte>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBinaryInteger<SByte>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<sbyte>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBinaryNumber<SByte>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static sbyte System.Numerics.IBinaryNumber<System.SByte>.AllBitsSet { get; }
```

Property Value

SByte

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBitwise Operators<SByte,SByte,SByte>.Bitwise And Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static sbyte IBitwiseOperators<sbyte,sbyte,sbyte>.operator & (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to `and` with `right`.

`right` SByte

The value to `and` with `left`.

Returns

SByte

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBitwiseOperators<SByte,SByte,SByte>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static sbyte IBitwiseOperators<sbyte,sbyte,sbyte>.operator | (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to `or` with `right`.

`right` SByte

The value to `or` with `left`.

Returns

SByte

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBitwiseOperators<SByte,SByte,SByte>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static sbyte IBitwiseOperators<sbyte,sbyte,sbyte>.operator ^ (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to xor with `right`.

`right` SByte

The value to xor with `left`.

Returns

SByte

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IBitwiseOperators<SByte,SByte,SByte>.OnesComplement(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static sbyte IBitwiseOperators<sbyte,sbyte,sbyte>.operator ~ (sbyte value);
```

Parameters

`value` SByte

The value for which to compute the ones-complement.

Returns

SByte

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IComparisonOperators<SByte,SByte,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<sbyte,sbyte,bool>.operator > (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IComparisonOperators<SByte,SByte,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<sbyte,sbyte,bool>.operator >= (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IComparisonOperators<SByte,SByte,Boolean>.Less Than Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<sbyte,sbyte,bool>.operator < (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IComparisonOperators<SByte,SByte,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<sbyte,sbyte,bool>.operator <= (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IDecrementOperators<SByte>.CheckedDecrement(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static sbyte IDecrementOperators<sbyte>.op_CheckedDecrement (sbyte value);
```

Parameters

`value` SByte

The value to decrement.

Returns

SByte

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IDecrementOperators<SByte>.Decrement(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static sbyte IDecrementOperators<sbyte>.operator -- (sbyte value);
```

Parameters

`value` SByte

The value to decrement.

Returns

SByte

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IDivisionOperators<SByte,SByte,SByte>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static sbyte IDivisionOperators<sbyte,sbyte,sbyte>.operator / (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value that `right` divides.

`right` SByte

The value that divides `left`.

Returns

SByte

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IEqualityOperators<SByte,SByte,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<sbyte,sbyte,bool>.operator == (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IEqualityOperators<SByte,SByte,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<sbyte,sbyte,bool>.operator != (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value to compare with `right`.

`right` SByte

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IIncrementOperators<SByte>.CheckedIncrement(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static sbyte IIncrementOperators<sbyte>.op_CheckedIncrement (sbyte value);
```

Parameters

`value` SByte

The value to increment.

Returns

SByte

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IIncrementOperators<SByte>.Increment(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static sbyte IIncrementOperators<sbyte>.operator ++ (sbyte value);
```

Parameters

`value` SByte

The value to increment.

Returns

SByte

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IMinMaxValue<SByte>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static sbyte System.Numerics.IMinMaxValue<System.SByte>.MaxValue { get; }
```

Property Value

SByte

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IMinMaxValue<SByte>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static sbyte System.Numerics.IMinMaxValue<System.SByte>.MinValue { get; }
```

Property Value

SByte

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IModulusOperators<SByte,SByte,SByte>.Modulus(SByte, SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static sbyte IModulusOperators<sbyte,sbyte,sbyte>.operator % (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value that `right` divides.

`right` SByte

The value that divides `left`.

Returns

SByte

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IMultiplicativeIdentity<SByte,SByte>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static sbyte
System.Numerics.IMultiplicativeIdentity<System.SByte,System.SByte>.MultiplicativeIdentity { get; }
```

Property Value

SByte

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.IMultiplyOperators<SByte,SByte,SByte>.CheckedMultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static sbyte IMultiplyOperators<sbyte,sbyte,sbyte>.op_CheckedMultiply (sbyte
left, sbyte right);
```

Parameters

`left` SByte

The value that `right` multiplies.

`right` SByte

The value that multiplies `left`.

Returns

SByte

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IMultiplyOperators<SByte,SByte,SByte>.MultiplyOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static sbyte IMultiplyOperators<sbyte,sbyte,sbyte>.operator * (sbyte left,
sbyte right);
```

Parameters

`left` SByte

The value that `right` multiplies.

`right` SByte

The value that multiplies `left`.

Returns

SByte

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumber<SByte>.MaxNumber(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

C#

```
static sbyte INumber<sbyte>.MaxNumber (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumber<SByte>.MinNumber(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static sbyte INumber<sbyte>.MinNumber (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsCanonical(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<sbyte>.IsCanonical (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsComplexNumber(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<sbyte>.IsComplexNumber (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsFinite(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<sbyte>.IsFinite (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsImaginaryNumber(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<sbyte>.IsImaginaryNumber (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsInfinity(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<sbyte>.IsInfinity (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsInteger(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<sbyte>.IsInteger (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsNaN(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<sbyte>.IsNaN (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsNegativeInfinity(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

```
C#
static bool INumberBase<sbyte>.IsNegativeInfinity (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsNormal(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<sbyte>.IsNormal (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsPositiveInfinity(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<sbyte>.IsPositiveInfinity (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsRealNumber(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<sbyte>.IsRealNumber (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsSubnormal(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<sbyte>.IsSubnormal (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.IsZero(SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<sbyte>.IsZero (sbyte value);
```

Parameters

`value` SByte

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.MaxMagnitudeNumber(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static sbyte INumberBase<sbyte>.MaxMagnitudeNumber (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.MinMagnitudeNumber(SByte, SByte) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static sbyte INumberBase<sbyte>.MinMagnitudeNumber (sbyte x, sbyte y);
```

Parameters

x SByte

The value to compare with `y`.

y SByte

The value to compare with `x`.

Returns

SByte

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static sbyte System.Numerics.INumberBase<System.SByte>.One { get; }
```

Property Value

SByte

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

SByte.INumberBase<SByte>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.SByte>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<sbyte>.TryConvertFromChecked<TOther> (TOther value,
out sbyte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` SByte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.INumberBase<SByte>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<sbyte>.TryConvertFromSaturating<TOther> (TOther
value, out sbyte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` SByte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.INumberBase<SByte>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<sbyte>.TryConvertFromTruncating<TOther> (TOther
value, out sbyte result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` SByte

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

SByte.INumberBase<SByte>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<sbyte>.TryConvertToChecked<TOther> (sbyte value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` SByte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SByte.INumberBase<SByte>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<sbyte>.TryConvertToSaturating<TOther> (sbyte value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` SByte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<sbyte>.TryConvertToTruncating<TOther> (sbyte value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` SByte

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.INumberBase<SByte>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static sbyte System.Numerics.INumberBase<System.SByte>.Zero { get; }
```

Property Value

SByte

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

SByte.IShift Operators<SByte,Int32,SByte>.Left Shift(SByte, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static sbyte IShiftOperators<sbyte,int,sbyte>.operator << (sbyte value, int
shiftAmount);
```

Parameters

`value` SByte

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

SByte

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IShift Operators<SByte,Int32,SByte>.Right Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static sbyte IShiftOperators<sbyte,int,sbyte>.operator >> (sbyte value, int
shiftAmount);
```

Parameters

`value` SByte

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

SByte

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IShift Operators<SByte,Int32,SByte>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static sbyte IShiftOperators<sbyte,int,sbyte>.op_UnsignedRightShift (sbyte
value, int shiftAmount);
```

Parameters

`value` SByte

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

SByte

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.ISignedNumber<SByte>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

```
C#

static sbyte System.Numerics.ISignedNumber<System.SByte>.NegativeOne { get;
}
```

Property Value

SByte

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

SByte.ISubtractionOperators<SByte,SByte,SByte>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static sbyte ISubtractionOperators<sbyte,sbyte,sbyte>.op_CheckedSubtraction
(sbyte left, sbyte right);
```

Parameters

`left` SByte

The value from which `right` is subtracted.

`right` SByte

The value that's subtracted from `left`.

Returns

SByte

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.ISubtractionOperators<SByte,SByte,SByte>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static sbyte ISubtractionOperators<sbyte,sbyte,sbyte>.operator - (sbyte
left, sbyte right);
```

Parameters

`left` SByte

The value from which `right` is subtracted.

`right` SByte

The value that's subtracted from `left`.

Returns

SByte

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IUnaryNegationOperators<SByte,SByte>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static sbyte IUnaryNegationOperators<sbyte,sbyte>.op_CheckedUnaryNegation
(sbyte value);
```

Parameters

`value` SByte

The value for which to compute the unary negation.

Returns

SByte

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IUnaryNegationOperators<SByte,SByte>.UnaryNegation(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static sbyte IUnaryNegationOperators<sbyte,sbyte>.operator - (sbyte value);
```

Parameters

`value` SByte

The value for which to compute the unary negation.

Returns

SByte

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SByte.IUnaryPlusOperators<SByte,SByte>.UnaryPlus(SByte) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static sbyte IUnaryPlusOperators<sbyte,sbyte>.operator + (sbyte value);
```

Parameters

`value` SByte

The value for which to compute the unary plus.

Returns

SByte

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SequencePosition Struct

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Represents a position in a non-contiguous set of memory. Properties of this type should not be interpreted by anything but the type that created it.

C#

```
public readonly struct SequencePosition : IEquatable<SequencePosition>
```

Inheritance Object → ValueType → SequencePosition

Implements IEquatable<SequencePosition>

Constructors

Expand table

SequencePosition(Object, Int32)	Initializes a new instance of the SequencePosition struct.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current instance is equal to another object.
Equals(SequencePosition)	Indicates whether the current instance is equal to another SequencePosition.
GetHashCode()	Returns the hash code for this instance.
GetInteger()	Returns the integer part of this SequencePosition.
GetObject()	Returns the object part of this SequencePosition.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SequencePosition(Object, Int32) Constructor

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Initializes a new instance of the SequencePosition struct.

```
C#
public SequencePosition (object? object, int integer);
```

Parameters

`object` Object

A non-contiguous set of memory.

`integer` Int32

The position in `object`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

SequencePosition.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current instance is equal to another object.
Equals(SequencePosition)	Indicates whether the current instance is equal to another SequencePosition.

Equals(Object)

Returns a value that indicates whether the current instance is equal to another object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with the current instance.

Returns

Boolean

`true` if `obj` is of type SequencePosition and is equal to the current instance; otherwise, `false`.

Remarks

Equality does not guarantee that the two instances point to the same location in a ReadOnlySequence<T>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Equals(SequencePosition)

Indicates whether the current instance is equal to another SequencePosition.

C#

```
public bool Equals (SequencePosition other);
```

Parameters

`other` SequencePosition

The sequence position to compare with the current instance.

Returns

Boolean

`true` if the two instances are equal; `false` otherwise.

Implements

Equals(T)

Remarks

Equality does not guarantee that the two instances point to the same location in a ReadOnlySequence<T>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SequencePosition.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for this instance.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SequencePosition.GetInteger Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Returns the integer part of this SequencePosition.

C#

```
public int GetInteger ();
```

Returns

Int32

The integer part of this sequence position.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SequencePosition.GetObject Method

Reference

Definition

Namespace: System

Assembly: System.Memory.dll

Returns the object part of this SequencePosition.

C#

```
public object? GetObject ();
```

Returns

Object

The object part of this sequence position.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SerializableAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that a class can be serialized using binary or XML serialization. This class cannot be inherited.

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
System.AttributeTargets.Delegate | System.AttributeTargets.Enum |
System.AttributeTargets.Struct, Inherited=false)]
public sealed class SerializableAttribute : Attribute
```

Inheritance Object → Attribute → SerializableAttribute

Attributes AttributeUsageAttribute

Examples

The following example demonstrates SOAP serialization of an object that's marked with the SerializableAttribute attribute.

C#

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

public class Test {
   public static void Main()  {

      // Creates a new TestSimpleObject object.
      TestSimpleObject obj = new TestSimpleObject();

      Console.WriteLine("Before serialization the object contains: ");
      obj.Print();

      // Opens a file and serializes the object into it in binary format.
      Stream stream = File.Open("data.xml", FileMode.Create);
```

```
        SoapFormatter formatter = new SoapFormatter();

        formatter.Serialize(stream, obj);
        stream.Close();

        // Empties obj.
        obj = null;

        // Opens file "data.xml" and deserializes the object from it.
        stream = File.Open("data.xml", FileMode.Open);
        formatter = new SoapFormatter();

        obj = (TestSimpleObject)formatter.Deserialize(stream);
        stream.Close();

        Console.WriteLine("");
        Console.WriteLine("After deserialization the object contains: ");
        obj.Print();
    }
}

// A test object that needs to be serialized.
[Serializable()]
public class TestSimpleObject  {

    public int member1;
    public string member2;
    public string member3;
    public double member4;

    // A field that is not serialized.
    [NonSerialized()] public string member5;

    public TestSimpleObject() {

        member1 = 11;
        member2 = "hello";
        member3 = "hello";
        member4 = 3.14159265;
        member5 = "hello world!";
    }

    public void Print() {

        Console.WriteLine("member1 = '{0}'", member1);
        Console.WriteLine("member2 = '{0}'", member2);
        Console.WriteLine("member3 = '{0}'", member3);
        Console.WriteLine("member4 = '{0}'", member4);
        Console.WriteLine("member5 = '{0}'", member5);
    }
}
```

Remarks

Apply the SerializableAttribute attribute to a type to indicate that instances of this type can be serialized using binary or XML serialization. The common language runtime throws SerializationException if any type in the graph of objects being serialized does not have the SerializableAttribute attribute applied.

Apply the SerializableAttribute attribute even if the class also implements the ISerializable interface to control the binary serialization process.

When you apply the SerializableAttribute attribute to a type, all private and public fields are serialized by default. You can control binary serialization more granularly by implementing the ISerializable interface to override the serialization process.

Or you can exclude fields from serialization by applying the NonSerializedAttribute attribute to the field. If a field of a binary-serializable type contains a pointer, a handle, or some other data structure that is specific to a particular environment, and cannot be meaningfully reconstituted in a different environment, then you might want to apply the NonSerializedAttribute attribute to that field.

For more information about using attributes, see Attributes. For more information about binary serialization, see System.Runtime.Serialization.

> **Note**
>
> This attribute does not apply to JSON serialization using **System.Text.Json**.

Constructors

Expand table

SerializableAttribute()	Initializes a new instance of the SerializableAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Attribute
- ISerializable
- NonSerializedAttribute
- Extending Metadata Using Attributes
- XML and SOAP Serialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

SerializableAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the SerializableAttribute class.

C#

```
public SerializableAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a single-precision floating-point number.

C#

```
public readonly struct Single : IComparable<float>, IConvertible,
IEquatable<float>, IParsable<float>, ISpanParsable<float>,
IUtf8SpanParsable<float>,
System.Numerics.IAdditionOperators<float,float,float>,
System.Numerics.IAdditiveIdentity<float,float>,
System.Numerics.IBinaryFloatingPointIeee754<float>,
System.Numerics.IBinaryNumber<float>,
System.Numerics.IBitwiseOperators<float,float,float>,
System.Numerics.IComparisonOperators<float,float,bool>,
System.Numerics.IDecrementOperators<float>,
System.Numerics.IDivisionOperators<float,float,float>,
System.Numerics.IEqualityOperators<float,float,bool>,
System.Numerics.IExponentialFunctions<float>,
System.Numerics.IFloatingPoint<float>,
System.Numerics.IFloatingPointConstants<float>,
System.Numerics.IFloatingPointIeee754<float>,
System.Numerics.IHyperbolicFunctions<float>,
System.Numerics.IIncrementOperators<float>,
System.Numerics.ILogarithmicFunctions<float>,
System.Numerics.IMinMaxValue<float>,
System.Numerics.IModulusOperators<float,float,float>,
System.Numerics.IMultiplicativeIdentity<float,float>,
System.Numerics.IMultiplyOperators<float,float,float>,
System.Numerics.INumber<float>, System.Numerics.INumberBase<float>,
System.Numerics.IPowerFunctions<float>,
System.Numerics.IRootFunctions<float>, System.Numerics.ISignedNumber<float>,
System.Numerics.ISubtractionOperators<float,float,float>,
System.Numerics.ITrigonometricFunctions<float>,
System.Numerics.IUnaryNegationOperators<float,float>,
System.Numerics.IUnaryPlusOperators<float,float>
```

Inheritance Object → ValueType → Single

Implements IComparable , IComparable<Single> , IConvertible , IEquatable<Single> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<Single> , IParsable<TSelf> , ISpanParsable<Single> , ISpanParsable<TSelf> , IAdditionOperators<Single,Single,Single> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<Single,Single> , IAdditiveIdentity<TSelf,TSelf> , IBinaryFloatingPointIeee754<Single> , IBinaryNumber<Single> , IBinaryNumber<TSelf> , IBitwiseOperators<Single,Single,Single> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<Single,Single,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<Single> , IDecrementOperators<TSelf> , IDivisionOperators<Single,Single,Single> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<Single,Single,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IExponentialFunctions<Single> , IExponentialFunctions<TSelf> , IFloatingPoint<Single> , IFloatingPoint<TSelf> , IFloatingPointConstants<Single> , IFloatingPointConstants<TSelf> , IFloatingPointIeee754<Single> , IFloatingPointIeee754<TSelf> , IHyperbolicFunctions<Single> , IHyperbolicFunctions<TSelf> , IIncrementOperators<Single> , IIncrementOperators<TSelf> , ILogarithmicFunctions<Single> , ILogarithmicFunctions<TSelf> , IMinMaxValue<Single> , IModulusOperators<Single,Single,Single> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<Single,Single> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<Single,Single,Single> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<Single> , INumber<TSelf> , INumberBase<Single> , INumberBase<TSelf> , IPowerFunctions<Single> , IPowerFunctions<TSelf> ,

IRootFunctions<Single> , IRootFunctions<TSelf> ,

ISignedNumber<Single> , ISignedNumber<TSelf> ,

ISubtractionOperators<Single,Single,Single> ,

ISubtractionOperators<TSelf,TSelf,TSelf> ,

ITrigonometricFunctions<Single> , ITrigonometricFunctions<TSelf> ,

IUnaryNegationOperators<Single,Single> ,

IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<Single,Single> , IUnaryPlusOperators<TSelf,TSelf> ,

IUtf8SpanFormattable , IUtf8SpanParsable<Single> ,

IUtf8SpanParsable<TSelf>

Remarks

For more information about this API, see Supplemental API remarks for Single.

Fields

Expand table

E	Represents the natural logarithmic base, specified by the constant, e.
Epsilon	Represents the smallest positive Single value that is greater than zero. This field is constant.
MaxValue	Represents the largest possible value of Single. This field is constant.
MinValue	Represents the smallest possible value of Single. This field is constant.
NaN	Represents not a number (`NaN`). This field is constant.
Negative Infinity	Represents negative infinity. This field is constant.
NegativeZero	Represents the number negative zero (-0).
Pi	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.
PositiveInfinity	Represents positive infinity. This field is constant.
Tau	Represents the number of radians in one turn, specified by the constant, τ.

Methods

Expand table

Abs(Single)	Computes the absolute of a value.
Acos(Single)	Computes the arc-cosine of a value.
Acosh(Single)	Computes the hyperbolic arc-cosine of a value.
AcosPi(Single)	Computes the arc-cosine of a value and divides the result by `pi`.
Asin(Single)	Computes the arc-sine of a value.
Asinh(Single)	Computes the hyperbolic arc-sine of a value.
AsinPi(Single)	Computes the arc-sine of a value and divides the result by `pi`.
Atan(Single)	Computes the arc-tangent of a value.
Atan2(Single, Single)	Computes the arc-tangent of the quotient of two values.
Atan2Pi(Single, Single)	Computes the arc-tangent for the quotient of two values and divides the result by `pi`.
Atanh(Single)	Computes the hyperbolic arc-tangent of a value.
AtanPi(Single)	Computes the arc-tangent of a value and divides the result by pi.
BitDecrement(Single)	Decrements a value to the smallest value that compares less than a given value.
BitIncrement(Single)	Increments a value to the smallest value that compares greater than a given value.
Cbrt(Single)	Computes the cube-root of a value.
Ceiling(Single)	Computes the ceiling of a value.
Clamp(Single, Single, Single)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.
CompareTo(Single)	Compares this instance to a specified single-precision floating-point number and returns an integer that indicates

	whether the value of this instance is less than, equal to, or greater than the value of the specified single-precision floating-point number.
CopySign(Single, Single)	Copies the sign of a value to the sign of another value.
Cos(Single)	Computes the cosine of a value.
Cosh(Single)	Computes the hyperbolic cosine of a value.
CosPi(Single)	Computes the cosine of a value that has been multipled by `pi`.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DegreesToRadians(Single)	Converts a given value from degrees to radians.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(Single)	Returns a value indicating whether this instance and a specified Single object represent the same value.
Exp(Single)	Computes `E` raised to a given power.
Exp10(Single)	Computes `10` raised to a given power.
Exp10M1(Single)	Computes `10` raised to a given power and subtracts one.
Exp2(Single)	Computes `2` raised to a given power.
Exp2M1(Single)	Computes `2` raised to a given power and subtracts one.
ExpM1(Single)	Computes `E` raised to a given power and subtracts one.
Floor(Single)	Computes the floor of a value.
FusedMultiplyAdd(Single, Single, Single)	Computes the fused multiply-add of three values.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type Single.

Hypot(Single, Single)	Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.
Ieee754Remainder(Single, Single)	Computes the remainder of two values as specified by IEEE 754.
ILogB(Single)	Computes the integer logarithm of a value.
IsEvenInteger(Single)	Determines if a value represents an even integral number.
IsFinite(Single)	Determines whether the specified value is finite (zero, subnormal or normal).
IsInfinity(Single)	Returns a value indicating whether the specified number evaluates to negative or positive infinity.
IsInteger(Single)	Determines if a value represents an integral value.
IsNaN(Single)	Returns a value that indicates whether the specified value is not a number (NaN).
IsNegative(Single)	Determines whether the specified value is negative.
IsNegativeInfinity(Single)	Returns a value indicating whether the specified number evaluates to negative infinity.
IsNormal(Single)	Determines whether the specified value is normal.
IsOddInteger(Single)	Determines if a value represents an odd integral number.
IsPositive(Single)	Determines if a value is positive.
IsPositiveInfinity(Single)	Returns a value indicating whether the specified number evaluates to positive infinity.
IsPow2(Single)	Determines if a value is a power of two.
IsRealNumber(Single)	Determines if a value represents a real number.
IsSubnormal(Single)	Determines whether the specified value is subnormal.
Lerp(Single, Single, Single)	Performs a linear interpolation between two values based on the given weight.
Log(Single)	Computes the natural (`base-E` logarithm of a value.
Log(Single, Single)	Computes the logarithm of a value in the specified base.
Log10(Single)	Computes the base-10 logarithm of a value.
Log10P1(Single)	Computes the base-10 logarithm of a value plus one.
Log2(Single)	Computes the log2 of a value.

Log2P1(Single)	Computes the base-2 logarithm of a value plus one.
LogP1(Single)	Computes the natural (`base-E`) logarithm of a value plus one.
Max(Single, Single)	Compares two values to compute which is greater.
MaxMagnitude(Single, Single)	Compares two values to compute which is greater.
MaxMagnitudeNumber(Single, Single)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
MaxNumber(Single, Single)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
Min(Single, Single)	Compares two values to compute which is lesser.
MinMagnitude(Single, Single)	Compares two values to compute which is lesser.
MinMagnitudeNumber(Single, Single)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
MinNumber(Single, Single)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Parse(String)	Converts the string representation of a number to its single-precision floating-point number equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.
Parse(String, NumberStyles, IFormat Provider)	Converts the string representation of a number in a specified style and culture-specific format to its single-

	precision floating-point number equivalent.
Pow(Single, Single)	Computes a value raised to a given power.
RadiansToDegrees(Single)	Converts a given value from radians to degrees.
ReciprocalEstimate(Single)	Computes an estimate of the reciprocal of a value.
ReciprocalSqrtEstimate(Single)	Computes an estimate of the reciprocal square root of a value.
RootN(Single, Int32)	Computes the n-th root of a value.
Round(Single)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Single, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Single, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Single, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
ScaleB(Single, Int32)	Computes the product of a value and its base-radix raised to the specified power.
Sign(Single)	Computes the sign of a value.
Sin(Single)	Computes the sine of a value.
SinCos(Single)	Computes the sine and cosine of a value.
SinCosPi(Single)	Computes the sine and cosine of a value.
Sinh(Single)	Computes the hyperbolic sine of a value.
SinPi(Single)	Computes the sine of a value that has been multiplied by `pi`.
Sqrt(Single)	Computes the square-root of a value.
Tan(Single)	Computes the tangent of a value.
Tanh(Single)	Computes the hyperbolic tangent of a value.
TanPi(Single)	Computes the tangent of a value that has been multipled by `pi`.
ToString()	Converts the numeric value of this instance to its equivalent string representation.

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
Truncate(Single)	Truncates a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current float number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Single)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Single)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Single)	Tries to convert a UTF-8 character span containing the string representation of a number to its single-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, Single)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Single)	Converts the span representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, Single)	Converts the string representation of a number in a character span to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, Single)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, Single)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(String, Single)	Converts the string representation of a number to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Operators

Expand table

Equality(Single, Single)	Returns a value that indicates whether two specified Single values are equal.
GreaterThan(Single, Single)	Returns a value that indicates whether a specified Single value is greater than another specified Single value.
GreaterThan OrEqual(Single, Single)	Returns a value that indicates whether a specified Single value is greater than or equal to another specified Single value.
Inequality(Single, Single)	Returns a value that indicates whether two specified Single values are not equal.
LessThan(Single, Single)	Returns a value that indicates whether a specified Single value is less than another specified Single value.
LessThanOrEqual(Single, Single)	Returns a value that indicates whether a specified Single value is less than or equal to another specified Single value.

Explicit Interface Implementations

Expand table

IAddition Operators<Single,Single,Single>.Addition(Single, Single)	Adds two values together to compute their sum.
IAdditiveIdentity<Single,Single>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryNumber<Single>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<Single,Single,Single>.Bitwise And(Single, Single)	Computes the bitwise-and of two values.
IBitwiseOperators<Single,Single,Single>.Bitwise Or(Single, Single)	Computes the bitwise-or of two values.

IBitwiseOperators<Single,Single,Single>.ExclusiveOr(Single, Single)	Computes the exclusive-or of two values.
IBitwiseOperators<Single,Single,Single>.OnesComplement(Single)	Computes the ones-complement representation of a given value.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).

IDecrementOperators<Single>.Decrement(Single)	Decrements a value.
IDivisionOperators<Single,Single,Single>.Division(Single, Single)	Divides one value by another to compute their quotient.
IFloatingPoint<Single>.GetExponentByteCount()	Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Single>.GetExponentShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current exponent.
IFloatingPoint<Single>.GetSignificandBitLength()	Gets the length, in bits, of the current significand.
IFloatingPoint<Single>.GetSignificandByteCount()	Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).
IFloatingPoint<Single>.TryWriteExponentBigEndian(Span<Byte>, Int32)	Tries to write the current exponent, in big-endian format, to a given span.
IFloatingPoint<Single>.TryWriteExponentLittleEndian(Span<Byte>, Int32)	Tries to write the current exponent, in little-endian format, to a given span.
IFloatingPoint<Single>.TryWriteSignificandBigEndian(Span<Byte>, Int32)	Tries to write the current significand, in big-endian format, to a given span.
IFloatingPoint<Single>.TryWriteSignificandLittleEndian(Span<Byte>, Int32)	Tries to write the current significand, in little-endian format, to a given span.
IFloatingPointConstants<Single>.E	Gets the mathematical constant `e`.
IFloatingPointConstants<Single>.Pi	Gets the mathematical constant `pi`.
IFloatingPointConstants<Single>.Tau	Gets the mathematical constant `tau`.
IFloatingPointIeee754<Single>.Epsilon	Gets the smallest value such that can be added to `0` that does not result in `0`.
IFloatingPointIeee754<Single>.NaN	Gets a value that represents `NaN`.
IFloatingPointIeee754<Single>.NegativeInfinity	Gets a value that represents negative `infinity`.
IFloatingPointIeee754<Single>.NegativeZero	Gets a value that represents negative `zero`.

IFloatingPointIeee754<Single>.PositiveInfinity	Gets a value that represents positive `infinity`.
IIncrementOperators<Single>.Increment(Single)	Increments a value.
IMinMaxValue<Single>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<Single>.MinValue	Gets the minimum value of the current type.
IModulus Operators<Single,Single,Single>.Modulus(Single, Single)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<Single,Single>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiply Operators<Single,Single,Single>.Multiply(Single, Single)	Multiplies two values together to compute their product.
INumberBase<Single>.IsCanonical(Single)	Determines if a value is in its canonical representation.
INumberBase<Single>.IsComplexNumber(Single)	Determines if a value represents a complex number.
INumberBase<Single>.IsImaginaryNumber(Single)	Determines if a value represents a pure imaginary number.
INumberBase<Single>.IsZero(Single)	Determines if a value is zero.
INumberBase<Single>.One	Gets the value `1` for the type.
INumberBase<Single>.Radix	Gets the radix, or base, for the type.
INumberBase<Single>.TryConvertFromChecked<TOther>(TOther, Single)	
INumberBase<Single>.TryConvertFromSaturating<TOther>(TOther, Single)	
INumberBase<Single>.TryConvertFromTruncating<TOther>(TOther, Single)	
INumberBase<Single>.TryConvertToChecked<TOther> (Single, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<Single>.TryConvertToSaturating<TOther> (Single, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

INumberBase<Single>.TryConvertToTruncating<TOther>(Single, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<Single>.Zero	Gets the value `0` for the type.
ISignedNumber<Single>.NegativeOne	Gets the value `-1` for the type.
ISubtractionOperators<Single,Single,Single>.Subtraction(Single, Single)	Subtracts two values to compute their difference.
IUnaryNegationOperators<Single,Single>.UnaryNegation(Single)	Computes the unary negation of a value.
IUnaryPlusOperators<Single,Single>.UnaryPlus(Single)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Decimal
- Double

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.E Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the natural logarithmic base, specified by the constant, e.

C#

```
public const float E = 2.71828175;
```

Field Value

Value* = *2.71828175

Single

Remarks

This is known as Euler's number and is approximately 2.7182818284590452354.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Epsilon Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest positive Single value that is greater than zero. This field is constant.

```
C#
public const float Epsilon = 1.401298E-45;
```

Field Value

Value = 1.401298E-45

Single

Remarks

For more information about this API, see Supplemental API remarks for Single.Epsilon.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of Single. This field is constant.

```
C#

public const float MaxValue = 3.40282347E+38;
```

Field Value

Value* = *3.40282347E+38

Single

Examples

The following code example demonstrates the MaxValue constant.

```
C#

if (s > Single.MaxValue)
{
    Console.WriteLine("Your number is larger than a Single.");
}
```

Remarks

The value of this constant is positive 3.40282347E+38.

The result of an operation that exceeds Single.MaxValue is Single.PositiveInfinity. In the following example, Single.PositiveInfinity results from addition, multiplication, and exponentiation operations when the result exceeds Single.MaxValue.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      float result1 = 1.867e38f + 2.385e38f;
      Console.WriteLine("{0} (Positive Infinity: {1})",
                        result1, Single.IsPositiveInfinity(result1));

      float result2 = 1.5935e25f * 7.948e20f;
      Console.WriteLine("{0} (Positive Infinity: {1})",
                        result2, Single.IsPositiveInfinity(result2));
   }
}
// The example displays the following output:
//    Infinity (Positive Infinity: True)
//    Infinity (Positive Infinity: True)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of Single. This field is constant.

```
C#

public const float MinValue = -3.40282347E+38;
```

Field Value

***Value* = -3.40282347E+38**

Single

Examples

The following code example demonstrates the MinValue constant.

```
C#

if (s < Single.MinValue)
{
    Console.WriteLine("Your number is smaller than a Single.");
}
```

Remarks

The value of this constant is negative 3.402823e38.

The result of an operation that is less than Single.MinValue is Single.NegativeInfinity. In the following example, Single.NegativeInfinity results from subtraction and multiplication operations when the result is less than Single.MinValue.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      float result1 = -8.997e37f + -2.985e38f;
      Console.WriteLine("{0} (Negative Infinity: {1})",
                        result1, Single.IsNegativeInfinity(result1));

      float result2 = -1.5935e25f * 7.948e32f;
      Console.WriteLine("{0} (Negative Infinity: {1})",
                        result2, Single.IsNegativeInfinity(result2));
   }
}
// The example displays the following output:
//    -Infinity (Negative Infinity: True)
//    -Infinity (Negative Infinity: True)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.NaN Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents not a number (`NaN`). This field is constant.

C#

```
public const float NaN = NaN;
```

Field Value

Value* = *NaN

Single

Examples

The following example demonstrates the NaN constant.

C#

```
Single zero = 0;

// This condition will return false.
if ((0 / zero) == Single.NaN)
{
    Console.WriteLine("0 / 0 can be tested with Single.NaN.");
}
else
{
    Console.WriteLine("0 / 0 cannot be tested with Single.NaN; use
Single.IsNan() instead.");
}
```

Remarks

A method or operator returns NaN when the result of an operation is undefined. For example, the result of dividing zero by zero is NaN, as the following example shows. (But note that dividing a non-zero number by zero returns either PositiveInfinity or NegativeInfinity, depending on the sign of the divisor.)

C#

```
float zero = 0.0f;
Console.WriteLine("{0} / {1} = {2}", zero, zero, zero/zero);
// The example displays the following output:
//         0 / 0 = NaN
```

In addition, a method call with a NaN value or an operation on a NaN value returns NaN, as the following example shows.

C#

```
float nan1 = Single.NaN;

Console.WriteLine("{0} + {1} = {2}", 3, nan1, 3 + nan1);
Console.WriteLine("Abs({0}) = {1}", nan1, Math.Abs(nan1));
// The example displays the following output:
//       3 + NaN = NaN
//       Abs(NaN) = NaN
```

Use the IsNaN method to determine whether a value is not a number. In general, Single operators cannot be used to compare Single.NaN with other Single values, although comparison methods (such as Equals and CompareTo) can. The following example illustrates the difference in behavior between Single comparison operators and methods.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Console.WriteLine("NaN == NaN: {0}", Single.NaN == Single.NaN);
      Console.WriteLine("NaN != NaN: {0}", Single.NaN != Single.NaN);
      Console.WriteLine("NaN.Equals(NaN): {0}",
Single.NaN.Equals(Single.NaN));
      Console.WriteLine("! NaN.Equals(NaN): {0}", !
Single.NaN.Equals(Single.NaN));
      Console.WriteLine("IsNaN: {0}", Double.IsNaN(Double.NaN));

      Console.WriteLine("\nNaN > NaN: {0}", Single.NaN > Single.NaN);
```

```
        Console.WriteLine("NaN >= NaN: {0}", Single.NaN >= Single.NaN);
        Console.WriteLine("NaN < NaN: {0}", Single.NaN < Single.NaN);
        Console.WriteLine("NaN < 100.0: {0}", Single.NaN < 100.0f);
        Console.WriteLine("NaN <= 100.0: {0}", Single.NaN <= 100.0f);
        Console.WriteLine("NaN >= 100.0: {0}", Single.NaN > 100.0f);
        Console.WriteLine("NaN.CompareTo(NaN): {0}",
Single.NaN.CompareTo(Single.NaN));
        Console.WriteLine("NaN.CompareTo(100.0): {0}",
Single.NaN.CompareTo(100.0f));
        Console.WriteLine("(100.0).CompareTo(Single.NaN): {0}",
(100.0f).CompareTo(Single.NaN));
    }
}
// The example displays the following output:
//       NaN == NaN: False
//       NaN != NaN: True
//       NaN.Equals(NaN): True
//       ! NaN.Equals(NaN): False
//       IsNaN: True
//
//       NaN > NaN: False
//       NaN >= NaN: False
//       NaN < NaN: False
//       NaN < 100.0: False
//       NaN <= 100.0: False
//       NaN >= 100.0: False
//       NaN.CompareTo(NaN): 0
//       NaN.CompareTo(100.0): -1
//       (100.0).CompareTo(Single.NaN): 1
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNaN(Single)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.NegativeInfinity Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents negative infinity. This field is constant.

C#

```
public const float NegativeInfinity = -Infinity;
```

Field Value

Value* = *-Infinity

Single

Examples

The following code example demonstrates the NegativeInfinity constant.

C#

```
// This will equal Infinity.
Console.WriteLine("10.0 minus NegativeInfinity equals {0}.", (10.0 -
Single.NegativeInfinity).ToString());
```

Remarks

The value of this constant is the result of dividing a negative number by zero.

This constant is returned when the result of an operation is less than MinValue.

Use IsNegativeInfinity to determine whether a value evaluates to negative infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNegativeInfinity(Single)
- IsInfinity(Single)
- PositiveInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.NegativeZero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number negative zero (-0).

C#

```
public const float NegativeZero = 0;
```

Field Value

Value* = *0

Single

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Pi Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π.

C#

```
public const float Pi = 3.14159274;
```

Field Value

Value = 3.14159274

Single

Remarks

Pi is approximately 3.1415926535897932385.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.PositiveInfinity Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents positive infinity. This field is constant.

C#

```
public const float PositiveInfinity = Infinity;
```

Field Value

Value* = *Infinity

Single

Examples

The following code example demonstrates the PositiveInfinity constant.

C#

```
// This will equal Infinity.
Console.WriteLine("PositiveInfinity plus 10.0 equals {0}.",
(Single.PositiveInfinity + 10.0).ToString());
```

Remarks

The value of this constant is the result of dividing a positive number by zero.

This constant is returned when the result of an operation is greater than MaxValue.

Use IsPositiveInfinity to determine whether a value evaluates to positive infinity.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsPositiveInfinity(Single)
- IsInfinity(Single)
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Tau Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of radians in one turn, specified by the constant, τ.

C#

```
public const float Tau = 6.28318548;
```

Field Value

Value* = *6.28318548

Single

Remarks

Tau is approximately 6.2831853071795864769.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Abs(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

C#

```
public static float Abs (float value);
```

Parameters

`value` Single

The value for which to get its absolute.

Returns

Single

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Acos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value.

```
C#
public static float Acos (float x);
```

Parameters

x Single

The value, in radians, whose arc-cosine is to be computed.

Returns

Single

The arc-cosine of `x`.

Implements

Acos(TSelf)

Remarks

This computes `arccos(x)` in the interval `[+0, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Acosh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-cosine of a value.

C#

```
public static float Acosh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic arc-cosine is to be computed.

Returns

Single

The hyperbolic arc-cosine of `x`.

Implements

Acosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.AcosPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-cosine of a value and divides the result by `pi`.

```
C#
public static float AcosPi (float x);
```

Parameters

x Single

The value whose arc-cosine is to be computed.

Returns

Single

The arc-cosine of `x`, divided by `pi`.

Implements

AcosPi(TSelf)

Remarks

This computes `arccos(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Asin(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value.

C#

```
public static float Asin (float x);
```

Parameters

x Single

The value, in radians, whose arc-sine is to be computed.

Returns

Single

The arc-sine of `x`.

Implements

Asin(TSelf)

Remarks

This computes `arcsin(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Asinh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-sine of a value.

C#

```
public static float Asinh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic arc-sine is to be computed.

Returns

Single

The hyperbolic arc-sine of `x`.

Implements

Asinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.AsinPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-sine of a value and divides the result by `pi`.

```
C#
public static float AsinPi (float x);
```

Parameters

x Single

The value whose arc-sine is to be computed.

Returns

Single

The arc-sine of `x`, divided by `pi`.

Implements

AsinPi(TSelf)

Remarks

This computes `arcsin(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Atan(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value.

C#

```
public static float Atan (float x);
```

Parameters

x Single

The value, in radians, whose arc-tangent is to be computed.

Returns

Single

The arc-tangent of `x`.

Implements

Atan(TSelf)

Remarks

This computes `arctan(x)` in the interval `[-π / 2, +π / 2]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Atan2(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of the quotient of two values.

C#

```
public static float Atan2 (float y, float x);
```

Parameters

y Single

The y-coordinate of a point.

x Single

The x-coordinate of a point.

Returns

Single

The arc-tangent of `y` divided-by `x`.

Implements

Atan2(TSelf, TSelf)

Remarks

This computes `arctan(y / x)` in the interval `[-π, +π]` radians.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Atan2Pi(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent for the quotient of two values and divides the result by `pi`.

```
C#
public static float Atan2Pi (float y, float x);
```

Parameters

y Single

The y-coordinate of a point.

x Single

The x-coordinate of a point.

Returns

Single

The arc-tangent of `y` divided-by `x`, divided by `pi`.

Implements

Atan2Pi(TSelf, TSelf)

Remarks

This computes `arctan(y / x) / π` in the interval `[-1, +1]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Atanh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic arc-tangent of a value.

C#

```
public static float Atanh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic arc-tangent is to be computed.

Returns

Single

The hyperbolic arc-tangent of `x`.

Implements

Atanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.AtanPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the arc-tangent of a value and divides the result by pi.

C#

```
public static float AtanPi (float x);
```

Parameters

x Single

The value whose arc-tangent is to be computed.

Returns

Single

The arc-tangent of `x`, divided by `pi`.

Implements

AtanPi(TSelf)

Remarks

This computes `arctan(x) / π` in the interval `[-0.5, +0.5]`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.BitDecrement(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value to the smallest value that compares less than a given value.

C#

```
public static float BitDecrement (float x);
```

Parameters

x Single

The value to be bitwise decremented.

Returns

Single

The smallest value that compares less than `x`.

Implements

BitDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.BitIncrement(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value to the smallest value that compares greater than a given value.

C#

```
public static float BitIncrement (float x);
```

Parameters

x Single

The value to be bitwise incremented.

Returns

Single

The smallest value that compares greater than `x`.

Implements

BitIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Cbrt(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cube-root of a value.

C#

```
public static float Cbrt (float x);
```

Parameters

x Single

The value whose cube-root is to be computed.

Returns

Single

The cube-root of `x`.

Implements

Cbrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Ceiling(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ceiling of a value.

C#

```
public static float Ceiling (float x);
```

Parameters

x Single

The value whose ceiling is to be computed.

Returns

Single

The ceiling of `x`.

Implements

Ceiling(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Clamp(Single, Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static float Clamp (float value, float min, float max);
```

Parameters

`value` Single

The value to clamp.

`min` Single

The inclusive minimum to which `value` should clamp.

`max` Single

The inclusive maximum to which `value` should clamp.

Returns

Single

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or to another Single instance and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object or the other Single instance.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.
CompareTo(Single)	Compares this instance to a specified single-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified single-precision floating-point number.

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified object.

C#

```
public int CompareTo (object? value);
```

Parameters

value Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`, or this instance is not a number (NaN) and `value` is a number.
Zero	This instance is equal to `value`, or this instance and value are both not a number (NaN), PositiveInfinity, or NegativeInfinity.
Greater than zero	This instance is greater than `value`, OR this instance is a number and `value` is not a number (NaN), OR `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a Single.

Examples

The following code example demonstrates the CompareTo method.

C#

```
obj1 = (Single)450;

if (a.CompareTo(obj1) < 0)
{
    Console.WriteLine("{0} is less than {1}.", a.ToString(),
obj1.ToString());
}

if (a.CompareTo(obj1) > 0)
{
    Console.WriteLine("{0} is greater than {1}.", a.ToString(),
obj1.ToString());
}
```

```
if (a.CompareTo(obj1) == 0)
{
    Console.WriteLine("{0} equals {1}.", a.ToString(), obj1.ToString());
}
```

Remarks

For more information about this API, see Supplemental API remarks for Single.CompareTo.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CompareTo(Single)

Compares this instance to a specified single-precision floating-point number and returns an integer that indicates whether the value of this instance is less than, equal to, or greater than the value of the specified single-precision floating-point number.

C#

```
public int CompareTo (float value);
```

Parameters

`value` Single

A single-precision floating-point number to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`, or this instance is not a number (NaN) and `value` is a number.
Zero	This instance is equal to `value`, or both this instance and `value` are not a number (NaN), PositiveInfinity, or NegativeInfinity.
Greater than zero	This instance is greater than `value`, or this instance is a number and `value` is not a number (NaN).

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";
```

```
    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
```

```
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
```

```
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

For more information about this API, see Supplemental API remarks for Single.CompareTo.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.CopySign(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

```
C#
public static float CopySign (float value, float sign);
```

Parameters

`value` Single

The value whose magnitude is used in the result.

`sign` Single

The value whose sign is used in the result.

Returns

Single

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Cos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value.

```
C#
public static float Cos (float x);
```

Parameters

x Single

The value, in radians, whose cosine is to be computed.

Returns

Single

The cosine of `x`.

Implements

Cos(TSelf)

Remarks

This computes `cos(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Cosh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic cosine of a value.

C#

```
public static float Cosh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic cosine is to be computed.

Returns

Single

The hyperbolic cosine of `x`.

Implements

Cosh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.CosPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the cosine of a value that has been multipled by `pi`.

```
C#
public static float CosPi (float x);
```

Parameters

x Single

The value, in half-revolutions, whose cosine is to be computed.

Returns

Single

The cosine of `x` multiplied-by `pi`.

Implements

CosPi(TSelf)

Remarks

This computes `cos(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static float CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Single

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static float CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Single

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

```
C#

public static float CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

Single

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.DegreesToRadians(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from degrees to radians.

```
C#
public static float DegreesToRadians (float degrees);
```

Parameters

degrees Single

The value to convert to radians.

Returns

Single

The value of `degrees` converted to radians.

Implements

DegreesToRadians(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two instances of Single represent the same value.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(Single)	Returns a value indicating whether this instance and a specified Single object represent the same value.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of Single and equals the value of this instance; otherwise, `false`.

Examples

The following code example demonstrates the Equals method.

```
C#

obj1 = (Single)500;
if (a.Equals(obj1)) {
    Console.WriteLine("The value type and reference type values are
equal.");
}
```

Remarks

The Equals method should be used with caution, because two apparently equivalent values can be unequal due to the differing precision of the two values. The following example reports that the Single value .3333 and the Single returned by dividing 1 by 3 are unequal.

```
C#

// Initialize two floats with apparently identical values
float float1 = .33333f;
object float2 = 1/3;
// Compare them for equality
Console.WriteLine(float1.Equals(float2));    // displays false
```

Rather than comparing for equality, one recommended technique involves defining an acceptable margin of difference between two values (such as .01% of one of the values). If the absolute value of the difference between the two values is less than or equal to that margin, the difference is likely to be due to differences in precision and, therefore, the values are likely to be equal. The following example uses this technique to compare .33333 and 1/3, the two Single values that the previous code example found to be unequal.

```
C#

// Initialize two floats with apparently identical values
float float1 = .33333f;
object float2 = (float) 1/3;
// Define the tolerance for variation in their values
float difference = Math.Abs(float1 * .0001f);

// Compare the values
// The output to the console indicates that the two values are equal
if (Math.Abs(float1 - (float) float2) <= difference)
```

```
    Console.WriteLine("float1 and float2 are equal.");
else
    Console.WriteLine("float1 and float2 are unequal.");
```

In this case, the values are equal.

> **Note**
>
> Because **Epsilon** defines the minimum expression of a positive value whose range is near zero, the margin of difference must be greater than **Epsilon**. Typically, it is many times greater than **Epsilon**.

The precision of floating-point numbers beyond the documented precision is specific to the implementation and version of the .NET Framework. Consequently, a comparison of two particular numbers might change between versions of the .NET Framework because the precision of the numbers' internal representation might change.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Single is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Single) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Single value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive numeric types except for Double in Visual Basic and except for Decimal and Double in C#, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Single) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
    static float value = 112;

    public static void Main()
    {
        byte byte1= 112;
```

```
        Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
        TestObjectForEquality(byte1);

        short short1 = 112;
        Console.WriteLine("value = short1: {0,16}", value.Equals(short1));
        TestObjectForEquality(short1);

        int int1 = 112;
        Console.WriteLine("value = int1: {0,18}", value.Equals(int1));
        TestObjectForEquality(int1);

        long long1 = 112;
        Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,16}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:              True
//       112 (Single) = 112 (Byte): False
//
//       value = short1:              True
//       112 (Single) = 112 (Int16): False
```

```
//
//       value = int1:                True
//       112 (Single) = 112 (Int32): False
//
//       value = long1:               True
//       112 (Single) = 112 (Int64): False
//
//       value = sbyte1:              True
//       112 (Single) = 112 (SByte): False
//
//       value = ushort1:              True
//       112 (Single) = 112 (UInt16): False
//
//       value = uint1:                True
//       112 (Single) = 112 (UInt32): False
//
//       value = ulong1:               True
//       112 (Single) = 112 (UInt64): False
//
//       value = dec1:                  False
//       112 (Single) = 112 (Decimal): False
//
//       value = dbl1:                 False
//       112 (Single) = 112 (Double): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Single)

Returns a value indicating whether this instance and a specified Single object represent the same value.

```
C#

public bool Equals (float obj);
```

Parameters

obj Single

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is equal to this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

For more information about this API, see Supplemental API remarks for Single.Equals.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(Object) method overloads. If an implicit conversion between the `obj` argument and a Single is defined and the argument is not typed as an Object, compilers may perform an implicit conversion and call the Equals(Single) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a Single value. The following example illustrates the difference in behavior between the two method overloads. In the case of all primitive numeric types except for Double in Visual Basic and except for Decimal and Double in C#, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(Single) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

```
C#
```

```
using System;

public class Example
{
   static float value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,16}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,18}", value.Equals(int1));
      TestObjectForEquality(int1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,17}", value.Equals(long1));
      TestObjectForEquality(long1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,16}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
      Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
      TestObjectForEquality(ushort1);

      uint uint1 = 112;
      Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
      TestObjectForEquality(uint1);

      ulong ulong1 = 112;
      Console.WriteLine("value = ulong1: {0,17}", value.Equals(ulong1));
      TestObjectForEquality(ulong1);

      decimal dec1 = 112m;
      Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
      TestObjectForEquality(dec1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }

   private static void TestObjectForEquality(Object obj)
   {
      Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                        value, value.GetType().Name,
```

```
                        obj, obj.GetType().Name,
                        value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (Single) = 112 (Byte): False
//
//       value = short1:             True
//       112 (Single) = 112 (Int16): False
//
//       value = int1:               True
//       112 (Single) = 112 (Int32): False
//
//       value = long1:              True
//       112 (Single) = 112 (Int64): False
//
//       value = sbyte1:             True
//       112 (Single) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (Single) = 112 (UInt16): False
//
//       value = uint1:               True
//       112 (Single) = 112 (UInt32): False
//
//       value = ulong1:              True
//       112 (Single) = 112 (UInt64): False
//
//       value = dec1:                 False
//       112 (Single) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (Single) = 112 (Double): False
```

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Exp(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power.

```
C#
public static float Exp (float x);
```

Parameters

x Single

The power to which `E` is raised.

Returns

Single

`E` raised to the power of `x`.

Implements

Exp(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Exp10(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power.

```
C#
public static float Exp10 (float x);
```

Parameters

x Single

The power to which `10` is raised.

Returns

Single

10^x

Implements

Exp10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Exp10M1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `10` raised to a given power and subtracts one.

```
C#
public static float Exp10M1 (float x);
```

Parameters

x Single

The power to which `10` is raised.

Returns

Single

`10ˣ - 1`

Implements

Exp10M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Exp2(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power.

```
C#
public static float Exp2 (float x);
```

Parameters

x Single

The power to which `2` is raised.

Returns

Single

2^x

Implements

Exp2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Exp2M1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `2` raised to a given power and subtracts one.

C#

```
public static float Exp2M1 (float x);
```

Parameters

x Single

The power to which `2` is raised.

Returns

Single

$2^x - 1$

Implements

Exp2M1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ExpM1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes `E` raised to a given power and subtracts one.

C#

```
public static float ExpM1 (float x);
```

Parameters

x Single

The power to which `E` is raised.

Returns

Single

$E^x - 1$

Implements

ExpM1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Floor(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the floor of a value.

C#

```
public static float Floor (float x);
```

Parameters

x Single

The value whose floor is to be computed.

Returns

Single

The floor of `x`.

Implements

Floor(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.FusedMultiplyAdd(Single, Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the fused multiply-add of three values.

```
C#

public static float FusedMultiplyAdd (float left, float right, float
addend);
```

Parameters

`left` Single

The value which `right` multiplies.

`right` Single

The value which multiplies `left`.

`addend` Single

The value that is added to the product of `left` and `right`.

Returns

Single

The result of `left` times `right` plus `addend` computed as one ternary operation.

Implements

FusedMultiplyAdd(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type Single.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, Single.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Hypot(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hypotenuse given two values representing the lengths of the shorter sides in a right-angled triangle.

```
C#
public static float Hypot (float x, float y);
```

Parameters

x Single

The value to square and add to `y`.

y Single

The value to square and add to `x`.

Returns

Single

The square root of `x`-squared plus `y`-squared.

Implements

Hypot(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Ieee754Remainder(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the remainder of two values as specified by IEEE 754.

C#

```
public static float Ieee754Remainder (float left, float right);
```

Parameters

`left` Single

The value which `right` divides.

`right` Single

The value which divides `left`.

Returns

Single

The remainder of `left` divided-by `right` as specified by IEEE 754.

Implements

Ieee754Remainder(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ILogB(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the integer logarithm of a value.

C#

```
public static int ILogB (float x);
```

Parameters

x Single

The value whose integer logarithm is to be computed.

Returns

Int32

The integer logarithm of `x`.

Implements

ILogB(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsEvenInteger(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsFinite(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is finite (zero, subnormal or normal).

C#

```
public static bool IsFinite (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if the specified value is finite (zero, subnormal or normal); otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsInfinity(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to negative or positive infinity.

```
C#

public static bool IsInfinity (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` evaluates to PositiveInfinity or NegativeInfinity; otherwise, `false`.

Implements

IsInfinity(TSelf)

Examples

The following code example demonstrates the IsInfinity method.

```
C#

// This will return "true".
Console.WriteLine("IsInfinity(3.0F / 0) == {0}.", Single.IsInfinity(3.0F /
0) ? "true" : "false");
```

Remarks

Floating-point operations return PositiveInfinity or NegativeInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsPositiveInfinity(Single)
- IsNegativeInfinity(Single)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IsInteger(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral value.

```
C#
public static bool IsInteger (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` and `3.0` will return `true` while `2.2` and `3.3` will return `false`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsNaN(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the specified value is not a number (NaN).

C#

```
public static bool IsNaN (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` evaluates to not a number (NaN); otherwise, `false`.

Implements

IsNaN(TSelf)

Examples

The following code example demonstrates the IsNaN method.

C#

```
// This will return true.
if (Single.IsNaN(0 / zero))
{
    Console.WriteLine("Single.IsNan() can determine whether a value is not-
a-number.");
}
```

Remarks

Floating-point operations return NaN to signal that result of the operation is undefined. For example, dividing 0.0 by 0.0 results in NaN.

> **Note**
>
> IsNaN returns `false` if a Single value is either PositiveInfinity or NegativeInfinity. To test for these values, use the IsInfinity, IsPositiveInfinity, and IsNegativeInfinity methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- NaN

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IsNegative(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is negative.

C#

```
public static bool IsNegative (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if negative, `false` otherwise.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsNegativeInfinity(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to negative infinity.

C#

```
public static bool IsNegativeInfinity (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` evaluates to NegativeInfinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Examples

The following code example demonstrates the IsNegativeInfinity method.

C#

```
// This will return true.
Console.WriteLine("IsNegativeInfinity(-5.0F / 0) == {0}.",
Single.IsNegativeInfinity(-5.0F / 0) ? "true" : "false");
```

Remarks

Floating-point operations return NegativeInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsInfinity(Single)
- IsPositiveInfinity(Single)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IsNormal(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is normal.

C#

```
public static bool IsNormal (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` is normal; `false` otherwise.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsOddInteger(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (float value);
```

Parameters

value Single

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsPositive(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
public static bool IsPositive (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Remarks

A return value of `false` does not imply that IsNegative(TSelf) will return `true`. A complex number, `a + bi` for non-zero `b`, is not positive or negative

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsPositiveInfinity(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether the specified number evaluates to positive infinity.

C#

```
public static bool IsPositiveInfinity (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` evaluates to PositiveInfinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Examples

The following code example demonstrates the IsPositiveInfinity method.

C#

```
// This will return true.
Console.WriteLine("IsPositiveInfinity(4.0F / 0) == {0}.",
Single.IsPositiveInfinity(4.0F / 0) ? "true" : "false");
```

Remarks

Floating-point operations return PositiveInfinity to signal an overflow condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsInfinity(Single)
- IsNegativeInfinity(Single)
- PositiveInfinity
- NegativeInfinity

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IsPow2(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsRealNumber(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
public static bool IsRealNumber (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Remarks

This function returns `true` for a complex number `a + bi` where `b` is zero.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IsSubnormal(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified value is subnormal.

C#

```
public static bool IsSubnormal (float f);
```

Parameters

`f` Single

A single-precision floating-point number.

Returns

Boolean

`true` if `f` is subnormal; `false` otherwise.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Lerp(Single, Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Performs a linear interpolation between two values based on the given weight.

C#

```
public static float Lerp (float value1, float value2, float amount);
```

Parameters

`value1` Single

The first value, which is intended to be the lower bound.

`value2` Single

The second value, which is intended to be the upper bound.

`amount` Single

A value, intended to be between 0 and 1, that indicates the weight of the interpolation.

Returns

Single

The interpolated value.

Implements

Lerp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Log Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Log(Single, Single)	Computes the logarithm of a value in the specified base.
Log(Single)	Computes the natural (`base-E` logarithm of a value.

Log(Single, Single)

Computes the logarithm of a value in the specified base.

C#

```
public static float Log (float x, float newBase);
```

Parameters

x Single

The value whose logarithm is to be computed.

newBase Single

The base in which the logarithm is to be computed.

Returns

Single

The base-`newBase` logarithm of `x`.

Implements

Log(TSelf, TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Log(Single)

Computes the natural (`base-E` logarithm of a value.

```
C#

public static float Log (float x);
```

Parameters

`x` Single

The value whose natural logarithm is to be computed.

Returns

Single

The natural logarithm of `x`.

Implements

Log(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Log10(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value.

C#

```
public static float Log10 (float x);
```

Parameters

x Single

The value whose base-10 logarithm is to be computed.

Returns

Single

The base-10 logarithm of `x`.

Implements

Log10(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Log10P1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-10 logarithm of a value plus one.

C#

```
public static float Log10P1 (float x);
```

Parameters

x Single

The value to which one is added before computing the base-10 logarithm.

Returns

Single

$\log_{10}(x + 1)$

Implements

Log10P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Log2(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static float Log2 (float value);
```

Parameters

`value` Single

The value whose log2 is to be computed.

Returns

Single

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Log2P1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the base-2 logarithm of a value plus one.

C#

```
public static float Log2P1 (float x);
```

Parameters

x Single

The value to which one is added before computing the base-2 logarithm.

Returns

Single

$\log_2(x + 1)$

Implements

Log2P1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.LogP1(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the natural (`base-E`) logarithm of a value plus one.

```
C#
public static float LogP1 (float x);
```

Parameters

x Single

The value to which one is added before computing the natural logarithm.

Returns

Single

$\log_e(x + 1)$

Implements

LogP1(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Max(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static float Max (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MaxMagnitude(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static float MaxMagnitude (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitude` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MaxMagnitudeNumber(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#
public static float MaxMagnitudeNumber (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `maximumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.MaxNumber(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
public static float MaxNumber (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0`

to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Min(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static float Min (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MinMagnitude(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
public static float MinMagnitude (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Remarks

For <see cref="T:System.Numerics.IFloatingPointIeee754`1> this method matches the IEEE 754:2019` minimumMagnitude`function. This requires NaN inputs to not be propagated back to the caller and for`-0.0`to be treated as less than`+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.MinMagnitudeNumber(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#
public static float MinMagnitudeNumber (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Remarks

For IFloatingPointIeee754<TSelf> this method matches the IEEE 754:2019 `minimumMagnitudeNumber` function. This requires NaN inputs to not be propagated back

to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.MinNumber(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
public static float MinNumber (float x, float y);
```

Parameters

x Single

The value to compare with `y`.

y Single

The value to compare with `x`.

Returns

Single

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimumNumber` function. This requires NaN inputs to not be propagated back to the caller and for `-0.0`

to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its single-precision floating-point number equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.
Parse(String)	Converts the string representation of a number to its single-precision floating-point number equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.

```
C#

public static float Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Single

A single-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a numeric value.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

Examples

The following code example uses the Parse(String, NumberStyles, IFormatProvider) method to parse the string representations of Single values. Each string in an array is parsed using the formatting conventions of the en-US, nl-NL, and a custom culture. The custom culture defines its group separator symbol as the underscore ("_") and its group size as two.

C#

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
      // Define an array of string values.
      string[] values = { " 987.654E-2", " 987,654E-2",  "(98765,43210)",
                          "9,876,543.210", "9.876.543,210",
"98_76_54_32,19" };
      // Create a custom culture based on the invariant culture.
      CultureInfo ci = new CultureInfo("");
      ci.NumberFormat.NumberGroupSizes = new int[] { 2 };
      ci.NumberFormat.NumberGroupSeparator = "_";

      // Define an array of format providers.
```

```
      CultureInfo[] providers = { new CultureInfo("en-US"),
                                  new CultureInfo("nl-NL"), ci };

      // Define an array of styles.
      NumberStyles[] styles = { NumberStyles.Currency, NumberStyles.Float
};

      // Iterate the array of format providers.
      foreach (CultureInfo provider in providers)
      {
         Console.WriteLine("Parsing using the {0} culture:",
                           provider.Name == String.Empty ? "Invariant" :
provider.Name);
         // Parse each element in the array of string values.
         foreach (string value in values)
         {
            foreach (NumberStyles style in styles)
            {
               try {
                  float number = Single.Parse(value, style, provider);
                  Console.WriteLine("   {0} ({1}) -> {2}",
                                    value, style, number);
               }
               catch (FormatException) {
                  Console.WriteLine("   '{0}' is invalid using {1}.",
value, style);
               }
               catch (OverflowException) {
                  Console.WriteLine("   '{0}' is out of the range of a
Single.", value);
               }
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
// Parsing using the en-US culture:
//    ' 987.654E-2' is invalid using Currency.
//     987.654E-2 (Float) -> 9.87654
//    ' 987,654E-2' is invalid using Currency.
//    ' 987,654E-2' is invalid using Float.
//    (98765,43210) (Currency) -> -9.876543E+09
//    '(98765,43210)' is invalid using Float.
//    9,876,543.210 (Currency) -> 9876543
//    '9,876,543.210' is invalid using Float.
//    '9.876.543,210' is invalid using Currency.
//    '9.876.543,210' is invalid using Float.
//    '98_76_54_32,19' is invalid using Currency.
//    '98_76_54_32,19' is invalid using Float.
//
// Parsing using the nl-NL culture:
//    ' 987.654E-2' is invalid using Currency.
//    ' 987.654E-2' is invalid using Float.
```

```
//     ' 987,654E-2' is invalid using Currency.
//      987,654E-2 (Float) -> 9.87654
//    (98765,43210) (Currency) -> -98765.43
//    '(98765,43210)' is invalid using Float.
//    '9,876,543.210' is invalid using Currency.
//    '9,876,543.210' is invalid using Float.
//    9.876.543,210 (Currency) -> 9876543
//    '9.876.543,210' is invalid using Float.
//    '98_76_54_32,19' is invalid using Currency.
//    '98_76_54_32,19' is invalid using Float.
//
// Parsing using the Invariant culture:
//    ' 987.654E-2' is invalid using Currency.
//     987.654E-2 (Float) -> 9.87654
//    ' 987,654E-2' is invalid using Currency.
//    ' 987,654E-2' is invalid using Float.
//    (98765,43210) (Currency) -> -9.876543E+09
//    '(98765,43210)' is invalid using Float.
//    9,876,543.210 (Currency) -> 9876543
//    '9,876,543.210' is invalid using Float.
//    '9.876.543,210' is invalid using Currency.
//    '9.876.543,210' is invalid using Float.
//    98_76_54_32,19 (Currency) -> 9.876543E+09
//    '98_76_54_32,19' is invalid using Float.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `style` parameter defines the style elements (such as white space, thousands separators, and currency symbols) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- AllowHexSpecifier
- HexNumber

The `s` parameter can contain NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol for the culture specified by `provider`. Depending on the value of `style`, it can also take the form:

[*ws*] [*$*] [*sign*][*integral-digits*,]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Elements framed in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	A negative sign symbol (-) or a positive sign symbol (+). The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
,	A culture-specific group separator. The current culture's group separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Single type. The remaining System.Globalization.NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in `s` in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation. This flag by itself supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (,) and decimal point (.) elements.

NumberStyles value	Elements permitted in `s` in addition to digits
Any	All elements. However, `s` cannot represent a hexadecimal number.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `value`. Typically, `provider` can be any one of the following:

- A CultureInfo object that represents the culture that provides numeric formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information.
- A NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat just returns itself.)
- A custom object that implements IFormatProvider and uses the GetFormat method to instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

If `s` is out of range of the Single data type, the method throws an OverflowException on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts a character span that contains the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent.

C#

```
public static float Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A character span that contains the number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Single

A single-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Exceptions

FormatException

`s` does not represent a numeric value.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

If `s` is out of range of the Single data type, the method returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static float Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.AllowThousands |
System.Globalization.NumberStyles.Float, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Single

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its single-precision floating-point number equivalent.

```
C#
public static float Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that contains a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

Single

A single-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to a Single value.

C#

```
protected void OkToSingle_Click(object sender, EventArgs e)
{
    string locale;
    float number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Single.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

This overload is typically used to convert text that can be formatted in a variety of ways to a Single value. For example, it can be used to convert the text entered by a user into an HTML text box to a numeric value.

The `s` parameter is interpreted using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. The `s` parameter can contain NumberFormatInfo.PositiveInfinitySymbol, NumberFormatInfo.NegativeInfinitySymbol, or NumberFormatInfo.NaNSymbol for the culture specified by `provider`, or it can contain a string of the form:

[*ws*][*sign*]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Optional elements are framed in square brackets ([and]). Elements that contain the term "digits" consist of a series of numeric characters ranging from 0 to 9.

Expand table

Element	Description
ws	A series of white-space characters.
sign	A negative sign symbol (-) or a positive sign symbol (+).
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Runs of *integral-digits* can be partitioned by a group-separator symbol. For example, in some cultures a comma (,) separates groups of thousands. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
.	A culture-specific decimal point symbol.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

For more information about numeric formats, see the Formatting Types topic.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific formatting information. When the Parse(String, IFormatProvider) method is invoked, it calls the `provider` parameter's GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information about the format of the `s` parameter. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null` or a NumberFormatInfo cannot be obtained, the formatting information for the current system culture is used.

If `s` is out of range of the Single data type, the method throws an OverflowException on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

Some examples of `s` are "100", "-123,456,789", "123.45e+6", "+500", "5e2", "3.1416", "600.", "-.123", and "-Infinity".

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String)

Converts the string representation of a number to its single-precision floating-point number equivalent.

C#

```
public static float Parse (string s);
```

Parameters

s String

A string that contains a number to convert.

Returns

Single

A single-precision floating-point number equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not represent a number in a valid format.

OverflowException
.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number less than Single.MinValue or greater than Single.MaxValue.

Examples

The following example uses the Parse(String) method to convert an array of strings to equivalent Single values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "100", "(100)", "-123,456,789", "123.45e+6",
                          "+500", "5e2", "3.1416", "600.", "-.123",
                          "-Infinity", "-1E-16",
Double.MaxValue.ToString(),
                          Single.MinValue.ToString(), String.Empty };
      foreach (string value in values)
      {
         try {
            float number = Single.Parse(value);
            Console.WriteLine("{0} -> {1}", value, number);
         }
         catch (FormatException) {
            Console.WriteLine("'{0}' is not in a valid format.", value);
         }
         catch (OverflowException) {
            Console.WriteLine("{0} is outside the range of a Single.",
value);
         }
      }
   }
}
// The example displays the following output:
//       100 -> 100
//       '(100)' is not in a valid format.
//       -123,456,789 -> -1.234568E+08
//       123.45e+6 -> 1.2345E+08
//       +500 -> 500
//       5e2 -> 500
//       3.1416 -> 3.1416
//       600. -> 600
//       -.123 -> -0.123
//       -Infinity -> -Infinity
```

```
//       -1E-16 -> -1E-16
//       1.79769313486232E+308 is outside the range of a Single.
//       -3.402823E+38 -> -3.402823E+38
//       '' is not in a valid format.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `s` parameter can contain the current culture's PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or a string of the form:

[*ws*][*sign*] [*integral-digits*[,]]*integral-digits*[.[*fractional-digits*]][e[*sign*]*exponential-digits*] [*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white space characters.
sign	A negative sign symbol or a positive sign symbol. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture. Only a leading sign can be used.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Runs of *integral-digits* can be partitioned by a group-separator symbol. For example, in some cultures a comma (,) separates groups of thousands. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.
,	A culture-specific thousands separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation.

Element	Description
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

The `s` parameter is interpreted using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use the Parse(String, NumberStyles) method overload.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see CurrentInfo. To parse a string by using the formatting information of a specific culture, use the Parse(String, IFormatProvider) or Parse(String, NumberStyles, IFormatProvider) method.

Ordinarily, if you pass the Parse method a string that is created by calling the ToString method, the original Single value is returned. However, because of a loss of precision, the values may not be equal.

If `s` is out of range of the Single data type, the method throws an OverflowException on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static float Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Single

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static float Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

Single

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its single-precision floating-point number equivalent.

```
C#

public static float Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string that contains a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Float combined with AllowThousands.

Returns

Single

A single-precision floating-point number that is equivalent to the numeric value or symbol specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not a number in a valid format.

OverflowException

.NET Framework and .NET Core 2.2 and earlier versions only: `s` represents a number that is less than Single.MinValue or greater than Single.MaxValue.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` includes the AllowHexSpecifier value.

Examples

The following example uses the Parse(String, NumberStyles) method to parse the string representations of Single values. The example uses formatting information for the en-US culture.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class ParseString
{
   public static void Main()
   {
      // Set current thread culture to en-US.
      Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("en-US");

      string value;
      NumberStyles styles;

      // Parse a string in exponential notation with only the
AllowExponent flag.
      value = "-1.063E-02";
      styles = NumberStyles.AllowExponent;
      ShowNumericValue(value, styles);

      // Parse a string in exponential notation
      // with the AllowExponent and Number flags.
      styles = NumberStyles.AllowExponent | NumberStyles.Number;
      ShowNumericValue(value, styles);

      // Parse a currency value with leading and trailing white space,
and
      // white space after the U.S. currency symbol.
      value = " $ 6,164.3299  ";
      styles = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
      ShowNumericValue(value, styles);

      // Parse negative value with thousands separator and decimal.
      value = "(4,320.64)";
      styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
                NumberStyles.Float;
      ShowNumericValue(value, styles);

      styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
```

```
                NumberStyles.Float | NumberStyles.AllowThousands;
        ShowNumericValue(value, styles);
    }

    private static void ShowNumericValue(string value, NumberStyles
styles)
    {
        Single number;
        try
        {
            number = Single.Parse(value, styles);
            Console.WriteLine("Converted '{0}' using {1} to {2}.",
                              value, styles.ToString(), number);
        }
        catch (FormatException)
        {
            Console.WriteLine("Unable to parse '{0}' with styles {1}.",
                              value, styles.ToString());
        }
        Console.WriteLine();
    }
}
// The example displays the following output to the console:
//    Unable to parse '-1.063E-02' with styles AllowExponent.
//
//    Converted '-1.063E-02' using AllowTrailingSign, AllowThousands,
Float to -0.01063.
//
//    Converted ' $ 6,164.3299  ' using Number, AllowCurrencySymbol to
6164.3299.
//
//    Unable to parse '(4,320.64)' with styles AllowTrailingSign,
AllowParentheses, Float.
//
//    Converted '(4,320.64)' using AllowTrailingSign, AllowParentheses,
AllowThousands, Float to -4320.64.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

The `style` parameter defines the style elements (such as white space, thousands separators, and currency symbols) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. The following NumberStyles members are not supported:

- AllowHexSpecifier

- HexNumber

The `s` parameter can contain the current culture's PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol. Depending on the value of `style`, it can also take the form:

[*ws*][*$*][*sign*][*integral-digits*[,]]*integral-digits*[.[*fractional-digits*]][E[*sign*]*exponential-digits*][*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

ws A series of white-space characters. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.

$ A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.

sign A negative sign symbol (-) or a positive sign symbol (+). The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.

integral-digits A series of digits ranging from 0 to 9 that specify the integral part of the number. The *integral-digits* element can be absent if the string contains the *fractional-digits* element.

, A culture-specific group separator. The current culture's group separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag

. A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.

fractional-digits A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.

E The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `value` parameter can represent a number in

exponential notation if `style` includes the NumberStyles.AllowExponent flag.

exponential-digits A series of digits ranging from 0 to 9 that specify an exponent.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Single type. The remaining System.Globalization.NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in `s` in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation. This flag by itself supports values in the form *digits*E*digits*; additional flags are needed to successfully parse strings with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The thousands separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the decimal point (.) symbol. The `s` parameter can also use

NumberStyles value	Elements permitted in `s` in addition to digits
	exponential notation.
Number	The `ws`, `sign`, thousands separator (,) and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

Some examples of `s` are "100", "-123,456,789", "123.45e+6", "+500", "5e2", "3.1416", "600.", "-.123", and "-Infinity".

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Parse(String, NumberStyles, IFormatProvider) overload.

Ordinarily, if you pass the Parse method a string that is created by calling the ToString method, the original Single value is returned. However, because of a loss of precision, the values may not be equal.

If `s` is out of range of the Single data type, the method throws an OverflowException on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Pow(Single, Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes a value raised to a given power.

```
C#
public static float Pow (float x, float y);
```

Parameters

x Single

The value which is raised to the power of `x`.

y Single

The power to which `x` is raised.

Returns

Single

`x` raised to the power of `y`.

Implements

Pow(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.RadiansToDegrees(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a given value from radians to degrees.

C#

```
public static float RadiansToDegrees (float radians);
```

Parameters

`radians` Single

The value to convert to degrees.

Returns

Single

The value of `radians` converted to degrees.

Implements

RadiansToDegrees(TSelf)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ReciprocalEstimate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal of a value.

```
C#
public static float ReciprocalEstimate (float x);
```

Parameters

x Single

The value whose estimate of the reciprocal is to be computed.

Returns

Single

An estimate of the reciprocal of `x`.

Implements

ReciprocalEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ReciprocalSqrtEstimate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes an estimate of the reciprocal square root of a value.

C#

```
public static float ReciprocalSqrtEstimate (float x);
```

Parameters

x Single

The value whose estimate of the reciprocal square root is to be computed.

Returns

Single

An estimate of the reciprocal square root of `x`.

Implements

ReciprocalSqrtEstimate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.RootN(Single, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the n-th root of a value.

```
C#
public static float RootN (float x, int n);
```

Parameters

x Single

The value whose `n`-th root is to be computed.

n Int32

The degree of the root to be computed.

Returns

Single

The `n`-th root of `x`.

Implements

RootN(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Round Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Round(Single)	Rounds a value to the nearest integer using the default rounding mode (ToEven).
Round(Single, Int32)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).
Round(Single, MidpointRounding)	Rounds a value to the nearest integer using the specified rounding mode.
Round(Single, Int32, MidpointRounding)	Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

Round(Single)

Rounds a value to the nearest integer using the default rounding mode (ToEven).

C#

```
public static float Round (float x);
```

Parameters

x Single

The value to round.

Returns

Single

The result of rounding `x` to the nearest integer using the default rounding mode.

Implements

Round(TSelf)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Single, Int32)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

C#

```
public static float Round (float x, int digits);
```

Parameters

`x` Single

The value to round.

`digits` Int32

The number of fractional digits to which `x` should be rounded.

Returns

Single

The result of rounding `x` to `digits` fractional-digits using the default rounding mode.

Implements

Round(TSelf, Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Single, MidpointRounding)

Rounds a value to the nearest integer using the specified rounding mode.

C#

```
public static float Round (float x, MidpointRounding mode);
```

Parameters

`x` Single

The value to round.

`mode` MidpointRounding

The mode under which `x` should be rounded.

Returns

Single

The result of rounding `x` to the nearest integer using `mode`.

Implements

Round(TSelf, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Round(Single, Int32, MidpointRounding)

Rounds a value to a specified number of fractional-digits using the default rounding mode (ToEven).

```
C#

public static float Round (float x, int digits, MidpointRounding mode);
```

Parameters

`x` Single

The value to round.

`digits` Int32

The number of fractional digits to which `x` should be rounded.

`mode` MidpointRounding

The mode under which `x` should be rounded.

Returns

Single

The result of rounding `x` to `digits` fractional-digits using `mode`.

Implements

Round(TSelf, Int32, MidpointRounding)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ScaleB(Single, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the product of a value and its base-radix raised to the specified power.

```
C#
public static float ScaleB (float x, int n);
```

Parameters

x Single

The value which base-radix raised to the power of `n` multiplies.

n Int32

The value to which base-radix is raised before multipliying `x`.

Returns

Single

The product of `x` and base-radix raised to the power of `n`.

Implements

ScaleB(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Sign(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (float value);
```

Parameters

`value` Single

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Sin(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value.

```
C#
public static float Sin (float x);
```

Parameters

x Single

The value, in radians, whose sine is to be computed.

Returns

Single

The sine of `x`.

Implements

Sin(TSelf)

Remarks

This computes `sin(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.SinCos(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value.

C#

```
public static (float Sin, float Cos) SinCos (float x);
```

Parameters

x Single

The value, in radians, whose sine and cosine are to be computed.

Returns

ValueTuple<Single,Single>

The sine and cosine of `x`.

Implements

SinCos(TSelf)

Remarks

This computes `(sin(x), cos(x))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.SinCosPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine and cosine of a value.

```
C#
public static (float SinPi, float CosPi) SinCosPi (float x);
```

Parameters

x Single

The value, in radians, whose sine and cosine are to be computed.

Returns

ValueTuple<Single,Single>

The sine and cosine of `x`.

Implements

SinCosPi(TSelf)

Remarks

This computes `(sin(x), cos(x))`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Sinh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic sine of a value.

C#

```
public static float Sinh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic sine is to be computed.

Returns

Single

The hyperbolic sine of `x`.

Implements

Sinh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.SinPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sine of a value that has been multiplied by `pi`.

C#

```
public static float SinPi (float x);
```

Parameters

x Single

The value, in half-revolutions, that is multipled by `pi` before computing its sine.

Returns

Single

The sine of `x` multiplied-by `pi`.

Implements

SinPi(TSelf)

Remarks

This computes `sin(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Sqrt(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the square-root of a value.

C#

```
public static float Sqrt (float x);
```

Parameters

x Single

The value whose square-root is to be computed.

Returns

Single

The square-root of `x`.

Implements

Sqrt(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Tan(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value.

```
C#
public static float Tan (float x);
```

Parameters

x Single

The value, in radians, whose tangent is to be computed.

Returns

Single

The tangent of `x`.

Implements

Tan(TSelf)

Remarks

This computes `tan(x)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.Tanh(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the hyperbolic tangent of a value.

C#

```
public static float Tanh (float x);
```

Parameters

x Single

The value, in radians, whose hyperbolic tangent is to be computed.

Returns

Single

The hyperbolic tangent of `x`.

Implements

Tanh(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.TanPi(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the tangent of a value that has been multipled by `pi`.

```
C#
public static float TanPi (float x);
```

Parameters

x Single

The value, in half-revolutions, that is multipled by `pi` before computing its tangent.

Returns

Single

The tangent of `x` multiplied-by `pi`.

Implements

TanPi(TSelf)

Remarks

This computes `tan(x * π)`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Examples

The following example uses the default Single.ToString method to display the string representations of a number of Single values.

C#

```
float number;

number = 1.6E20F;
// Displays 1.6E+20.
Console.WriteLine(number.ToString());

number = 1.6E2F;
// Displays 160.
Console.WriteLine(number.ToString());

number = -3.541F;
// Displays -3.541.
Console.WriteLine(number.ToString());

number = -1502345222199E-07F;
// Displays -150234.5222199.
Console.WriteLine(number.ToString());

number = -150234522219901995 74E-09F;
// Displays -15023452221.9902.
Console.WriteLine(number.ToString());

number = .60344F;
// Displays 0.60344.
Console.WriteLine(number.ToString());

number = .000000001F;
// Displays 1E-09.
Console.WriteLine(number.ToString());
```

The following code example illustrates the use of the Parse(String) method along with the ToString() method.

C#

```
bool done = false;
string inp;
do
{
    Console.Write("Enter a real number: ");
    inp = Console.ReadLine();
    try
    {
        s = Single.Parse(inp);
```

```
        Console.WriteLine("You entered {0}.", s.ToString());
        done = true;
    }
    catch (FormatException)
    {
        Console.WriteLine("You did not enter a number.");
    }
    catch (Exception e)
    {
        Console.WriteLine("An exception occurred while parsing your
response: {0}", e.ToString());
    }
} while (!done);
```

Remarks

The ToString() method formats a Single value in the default ("G", or general) format of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or a string of the form:

[sign]integral-digits[.[fractional-digits]][e[sign]exponential-digits]

Optional elements are framed in square brackets ([and]). Elements that contain the term "digits" consist of a series of numeric characters ranging from 0 to 9. The following table lists each element:

Expand table

Element	Description
sign	A negative sign or positive sign symbol.
integral-digits	A series of digits specifying the integral part of the number. Integral-digits can be absent if there are fractional-digits.

Element	Description
'.'	A culture-specific decimal point symbol.
fractional-digits	A series of digits specifying the fractional part of the number.
'e'	A lowercase character 'e', indicating exponential (scientific) notation.
exponential-digits	A series of digits specifying an exponent.

Some examples of the return value are "100", "-123,456,789", "123.45e+6", "500", "3.1416", "600", "-0.123", and "-Infinity".

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

```
C#

public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Implements

ToString(IFormatProvider)

Examples

The following example displays the string representation of two Single values using CultureInfo objects that represent several different cultures.

```
C#

float value;

value = -16325.62015F;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));

value = 16034.125E21F;
// Display value using the invariant culture.
Console.WriteLine(value.ToString(CultureInfo.InvariantCulture));
// Display value using the en-GB culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("en-
```

```
GB")));
// Display value using the de-DE culture.
Console.WriteLine(value.ToString(CultureInfo.CreateSpecificCulture("de-
DE")));
// This example displays the following output to the console:
//       -16325.62015
//       -16325.62015
//       -16325,62015
//       1.6034125E+25
//       1.6034125E+25
//       1,6034125E+25
```

Remarks

The ToString(IFormatProvider) method formats a Single value in the default ("G", or general) format of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or a string of the form:

[sign]integral-digits[.[fractional-digits]][e[sign]exponential-digits]

Optional elements are framed in square brackets ([and]). Elements containing the term "digits" consist of a series of numeric characters ranging from 0 to 9. The following table lists each element.

Expand table

Element	Description
sign	A negative sign or positive sign symbol.
integral-digits	A series of digits specifying the integral part of the number. Integral-digits can be absent if there are fractional-digits.
'.'	A culture-specific decimal point symbol.

Element	Description
fractional-digits	A series of digits specifying the fractional part of the number.
'e'	A lowercase character 'e', indicating exponential (scientific) notation.
exponential-digits	A series of digits specifying an exponent.

Some examples of the return value are "100", "-123,456,789", "123.45e+6", "500", "3.1416", "600", "-0.123", and "-Infinity".

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a CultureInfo object or a NumberFormatInfo object. The `provider` parameter supplies culture-specific information used in formatting. If `provider` is `null`, the return value is formatted using the NumberFormatInfo data for the current culture.

To convert a Single value to its string representation using a specified culture and a specific format string, call the Single.ToString(String, IFormatProvider) method.

See also

- Parse(String)
- String
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid.

Examples

The following example defines a numeric value and formats it as a currency value by using the "C" standard numeric format string and as a numeric value to three decimal places by using the "N" standard numeric format string. The result strings are formatted by using the conventions of the en-US culture. For more information on numeric format strings, see Standard Numeric Format Strings and Custom Numeric Format Strings.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Double number = 1764.3789;

      // Format as a currency value.
      Console.WriteLine(number.ToString("C"));

      // Format as a numeric value with 3 decimal places.
      Console.WriteLine(number.ToString("N3"));
   }
}
// The example displays the following output:
//       $1,764.38
//       1,764.379
```

The following example displays several Single values using each of the supported standard numeric format specifiers together with two custom numeric format strings. One of those custom format strings illustrates how to pad a Single value with leading zeros. In converting the numeric values to strings, the example uses the formatting conventions of the en-US culture.

C#

```
float[] numbers= { 1054.32179F, -195489100.8377F, 1.0437E21F,
                   -1.0573e-05F };
string[] specifiers = { "C", "E", "e", "F", "G", "N", "P",
                        "R", "#,000.000", "0.###E-000",
                        "000,000,000,000.00###" };

foreach (float number in numbers)
{
   Console.WriteLine("Formatting of {0}:", number);
   foreach (string specifier in specifiers)
      Console.WriteLine("   {0,5}: {1}",
                        specifier, number.ToString(specifier));

   Console.WriteLine();
}
// The example displays the following output to the console:
//       Formatting of 1054.32179:
//              C: $1,054.32
//              E: 1.054322E+003
//              e: 1.054322e+003
//              F: 1054.32
//              G: 1054.32179
```

```
//              N: 1,054.32
//              P: 105,432.18 %
//              R: 1054.32179
//         #,000.000: 1,054.322
//         0.###E-000: 1.054E003
//         000,000,000,000.00###: 000,000,001,054.322
//
//       Formatting of -195489100.8377:
//              C: ($195,489,100.84)
//              E: -1.954891E+008
//              e: -1.954891e+008
//              F: -195489100.84
//              G: -195489100.8377
//              N: -195,489,100.84
//              P: -19,548,910,083.77 %
//              R: -195489100.8377
//         #,000.000: -195,489,100.838
//         0.###E-000: -1.955E008
//         000,000,000,000.00###: -000,195,489,100.00
//
//       Formatting of 1.0437E+21:
//              C: $1,043,700,000,000,000,000,000.00
//              E: 1.043700E+021
//              e: 1.043700e+021
//              F: 1043700000000000000000.00
//              G: 1.0437E+21
//              N: 1,043,700,000,000,000,000,000.00
//              P: 104,370,000,000,000,000,000,000.00 %
//              R: 1.0437E+21
//         #,000.000: 1,043,700,000,000,000,000,000.000
//         0.###E-000: 1.044E021
//         000,000,000,000.00###: 1,043,700,000,000,000,000,000.00
//
//       Formatting of -1.0573E-05:
//              C: $0.00
//              E: -1.057300E-005
//              e: -1.057300e-005
//              F: 0.00
//              G: -1.0573E-05
//              N: 0.00
//              P: 0.00 %
//              R: -1.0573E-05
//         #,000.000: 000.000
//         0.###E-000: -1.057E-005
//         000,000,000,000.00###: -000,000,000,000.00001
```

Remarks

The ToString(String) method formats a Single value in a specified format by using the conventions of the current culture. If you want to use the default ("G", or general)

format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or the string representation of the value of the current instance, as specified by `format`.

The `format` parameter can be any valid standard numeric format specifier except for D and X, as well as any combination of custom numeric format specifiers. If format is `null` or an empty string, the return value is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

By default, the return value only contains 7 digits of precision although a maximum of 9 digits is maintained internally. If the value of this instance has greater than 7 digits, ToString(String) returns PositiveInfinitySymbol or NegativeInfinitySymbol instead of the expected number. If you require more precision, specify `format` with the "G9" format specification, which always returns 9 digits of precision, or "R", which returns 7 digits if the number can be represented with that precision or 9 digits if the number can only be represented with maximum precision.

See also

- Parse(String)
- String
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

```
C#

public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Examples

The following example displays a Single value using each of the supported standard numeric format specifiers for several different cultures.

C#

```
float value = 16325.62901F;
string specifier;
CultureInfo culture;

// Use standard numeric format specifiers.
specifier = "G";
culture = CultureInfo.CreateSpecificCulture("eu-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,62901
Console.WriteLine(value.ToString(specifier,
CultureInfo.InvariantCulture));
// Displays:    16325.62901

specifier = "C";
culture = CultureInfo.CreateSpecificCulture("en-US");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    $16,325.63
culture = CultureInfo.CreateSpecificCulture("en-GB");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    £16,325.63

specifier = "E04";
culture = CultureInfo.CreateSpecificCulture("sv-SE");
Console.WriteLine(value.ToString(specifier, culture));
// Displays: 1,6326E+004
 culture = CultureInfo.CreateSpecificCulture("en-NZ");
 Console.WriteLine(value.ToString(specifier, culture));
// Displays:    1.6326E+004

specifier = "F";
culture = CultureInfo.CreateSpecificCulture("fr-FR");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325,63
culture = CultureInfo.CreateSpecificCulture("en-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16325.63

specifier = "N";
culture = CultureInfo.CreateSpecificCulture("es-ES");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16.325,63
culture = CultureInfo.CreateSpecificCulture("fr-CA");
Console.WriteLine(value.ToString(specifier, culture));
// Displays:    16 325,63

specifier = "P";
```

```
culture = CultureInfo.InvariantCulture;
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.26 %
culture = CultureInfo.CreateSpecificCulture("ar-EG");
Console.WriteLine((value/10000).ToString(specifier, culture));
// Displays:    163.256 %
```

Remarks

The ToString(String, IFormatProvider) method formats a Single value in a specified format of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The return value can be PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol, or the string representation of the value of the current instance, as specified by `format`.

The `format` parameter can be any valid standard numeric format specifier except for D and X, as well as any combination of custom numeric format specifiers. If `format` is `null` or an empty string, the return value for this instance is formatted with the general numeric format specifier ("G").

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object. Typically, `provider` is a CultureInfo object or a NumberFormatInfo object. The `provider` parameter supplies culture-

specific information used in formatting. If `provider` is `null`, the return value is formatted with the NumberFormatInfo object for the current culture.

By default, the return value only contains 7 digits of precision although a maximum of 9 digits is maintained internally. If the value of this instance has greater than 7 digits, ToString returns PositiveInfinitySymbol or NegativeInfinitySymbol instead of the expected number. If you require more precision, specify `format` with the "G9" format specification, which always returns 9 digits of precision, or "R", which returns 7 digits if the number can be represented with that precision or 9 digits if the number can only be represented with maximum precision.

See also

- Parse(String)
- String
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.Truncate(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Truncates a value.

```
C#
public static float Truncate (float x);
```

Parameters

x Single

The value to truncate.

Returns

Single

The truncation of `x`.

Implements

Truncate(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current float number instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current float number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the charactes that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

```
C#
```

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, IFormatProvider, Single)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, Single)	Converts the string representation of a number in a character span to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, Single)	Converts the string representation of a number to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, Single)	Tries to parse a span of UTF-8 characters into a value.
TryParse(String, IFormatProvider, Single)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Single)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, Single)	Tries to convert a UTF-8 character span containing the string representation of a number to its single-precision floating-point number equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Single)	Converts the span representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return

	value indicates whether the conversion succeeded or failed.
TryParse(String, NumberStyles, IFormatProvider, Single)	Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

TryParse(ReadOnlySpan<Char>, IFormatProvider, Single)

Tries to parse a span of characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out float result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result Single

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, Single)

Converts the string representation of a number in a character span to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, out float result);
```

Parameters

`s` ReadOnlySpan<Char>

>A character span that contains the string representation of the number to convert.

`result` Single

When this method returns, contains the single-precision floating-point number equivalent of the `s` parameter, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty or is not a number in a valid format. If `s` is a valid number less than Single.MinValue, `result` is NegativeInfinity. If `s` is a valid number greater than Single.MaxValue, `result` is PositiveInfinity. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, Single)

Converts the string representation of a number to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, out float result);
```

Parameters

`s` String

A string representing a number to convert.

`result` Single

When this method returns, contains single-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty or is not a number in a valid format. It also fails on .NET Framework and .NET Core 2.2 and earlier versions if `s` represents a number less than Single.MinValue or greater than Single.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example uses the TryParse(String, Single) method to convert the string representations of numeric values to Single values. It assumes that en-US is the current culture.

C#

```
string value;
float number;

// Parse a floating-point value with a thousands separator.
value = "1,643.57";
if (Single.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);

// Parse a floating-point value with a currency symbol and a
// thousands separator.
value = "$1,643.57";
if (Single.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);

// Parse value in exponential notation.
value = "-1.643e6";
if (Single.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);

// Parse a negative integer value.
value = "-168934617882109132";
if (Single.TryParse(value, out number))
   Console.WriteLine(number);
else
   Console.WriteLine("Unable to parse '{0}'.", value);
// The example displays the following output:
//       1643.57
//       Unable to parse '$1,643.57'.
//       -164300
//       -1.689346E+17
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

This overload differs from the Single.Parse(String) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed numeric value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `s` parameter can contain PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol (the string comparison is case-sensitive), or a string of the form:

[ws][sign][integral-digits,]integral-digits[.[fractional-digits]][e[sign]exponential-digits][ws]

Elements in square brackets are optional. The following table describes each element.

Expand table

Element	Description
ws	A series of white-space characters.
sign	A negative sign or positive sign symbol.
integral-digits	A series of numeric characters ranging from 0 to 9 that specify the integral part of the number. Integral-digits can be absent if there are fractional-digits.
,	A culture-specific group separator symbol.
.	A culture-specific decimal point symbol.
fractional-digits	A series of numeric characters ranging from 0 to 9 that specify the fractional part of the number.
E	An uppercase or lowercase character 'e', that indicates exponential (scientific) notation.
exponential-digits	A series of numeric characters ranging from 0 to 9 that specify an exponent.

The `s` parameter is interpreted using a combination of the NumberStyles.Float and NumberStyles.AllowThousands flags. This means that white space and thousands separators are allowed but currency symbols are not. To explicitly define the elements (such as currency symbols, thousands separators, and white space) that can be present in `s`, use the TryParse(String, NumberStyles, IFormatProvider, Single) method overload.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string using the formatting information of some other specified culture, use the TryParse(String, NumberStyles, IFormatProvider, Single) method overload.

Ordinarily, if you pass the Single.TryParse method a string that is created by calling the Single.ToString method, the original Single value is returned. However, because of a loss of precision, the values may not be equal.

If `s` is out of range of the Single data type, the method returns `false` on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Byte>, IFormatProvider, Single)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out float result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Single

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, IFormatProvider, Single)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
float result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Single

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, Single)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
float result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` Single

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, Single)

Tries to convert a UTF-8 character span containing the string representation of a number to its single-precision floating-point number equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out float
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A read-only UTF-8 character span that contains the number to convert.

`result` Single

When this method returns, contains a single-precision floating-point number equivalent of the numeric value or symbol contained in `utf8Text` if the conversion succeeded or zero if the conversion failed. The conversion fails if the `utf8Text` is Empty or is not in a valid format. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Single)

Converts the span representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#

public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
float result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only character span that contains the number to convert. The span is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Single

When this method returns, contains the single-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, represents a number less than Single.MinValue or greater than Single.MaxValue, or if `style` is not a valid combination of NumberStyles enumerated constants. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, Single)

Converts the string representation of a number in a specified style and culture-specific format to its single-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out float result);
```

Parameters

`s` String

A string representing a number to convert.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Float combined with AllowThousands.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` Single

When this method returns, contains the single-precision floating-point number equivalent to the numeric value or symbol contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or if `style` is not a valid combination of NumberStyles enumeration constants. It also fails on .NET Framework or .NET Core 2.2 and earlier versions if `s` represents a number less than Single.MinValue or greater than Single.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is the AllowHexSpecifier value.

Examples

The following example demonstrates the use of the Single.TryParse(String, NumberStyles, IFormatProvider, Single) method to parse the string representation of numbers that have a particular style and are formatted using the conventions of a particular culture.

C#

```
string value;
System.Globalization.NumberStyles style;
System.Globalization.CultureInfo culture;
float number;

// Parse currency value using en-GB culture.
value = "£1,097.63";
style = System.Globalization.NumberStyles.Number |
        System.Globalization.NumberStyles.AllowCurrencySymbol;
culture = System.Globalization.CultureInfo.CreateSpecificCulture("en-
GB");
if (Single.TryParse(value, style, culture, out number))
```

```
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);

value = "1345,978";
style = System.Globalization.NumberStyles.AllowDecimalPoint;
culture = System.Globalization.CultureInfo.CreateSpecificCulture("fr-
FR");
if (Single.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);

value = "1.345,978";
style = System.Globalization.NumberStyles.AllowDecimalPoint |
        System.Globalization.NumberStyles.AllowThousands;
culture = System.Globalization.CultureInfo.CreateSpecificCulture("es-
ES");
if (Single.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);

value = "1 345,978";
if (Single.TryParse(value, style, culture, out number))
   Console.WriteLine("Converted '{0}' to {1}.", value, number);
else
   Console.WriteLine("Unable to convert '{0}'.", value);
// The example displays the following output:
//       Converted '£1,097.63' to 1097.63.
//       Converted '1345,978' to 1345.978.
//       Converted '1.345,978' to 1345.978.
//       Unable to convert '1 345,978'.
```

Remarks

In .NET Core 3.0 and later, values that are too large to represent are rounded to PositiveInfinity or NegativeInfinity as required by the IEEE 754 specification. In prior versions, including .NET Framework, parsing a value that was too large to represent resulted in failure.

This overload differs from the Parse(String, NumberStyles, IFormatProvider) method by returning a Boolean value that indicates whether the parse operation succeeded instead of returning the parsed numeric value. It eliminates the need to use exception handling to test for a FormatException in the event that `s` is invalid and cannot be successfully parsed.

The `style` parameter defines the allowable format of the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles

enumeration. The following NumberStyles members are not supported:

- NumberStyles.AllowHexSpecifier
- NumberStyles.HexNumber

The `s` parameter can contain PositiveInfinitySymbol, NegativeInfinitySymbol, NaNSymbol for the culture indicated by `provider`. In addition, depending on the value of `style`, the `s` parameter may include the following elements:

[ws] [$] [sign][integral-digits,]integral-digits[.fractional-digits][e[sign]exponential-digits][ws]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag. It can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern or NumberFormatInfo.CurrencyPositivePattern properties of the NumberFormatInfo object returned by the IFormatProvider.GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
integral-digits	A series of digits ranging from 0 to 9 that specify the integral part of the number. Integral-digits can be absent if there are fractional-digits.
,	A culture-specific thousands separator symbol. The current culture's thousands separator symbol can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.

Element	Description
fractional-digits	A series of digits ranging from 0 to 9 that specify the fractional part of the number. Fractional digits can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
e	The e or E character, which indicates that `s` can represent a number using exponential notation. The `s` parameter can represent a number in exponential notation if style includes the NumberStyles.AllowExponent flag.
exponential-digits	A series of digits ranging from 0 to 9 that specify an exponent.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the Single type. The remaining System.Globalization.NumberStyles members control elements that may be but are not required to be present in the input string. The following table indicates how individual NumberStyles flags affect the elements that may be present in `s`.

Expand table

NumberStyles value	Elements permitted in s in addition to digits
None	The *integral-digits* element only.
AllowDecimalPoint	The . and *fractional-digits* elements.
AllowExponent	The `s` parameter can also use exponential notation. This flag by itself supports values in the form *integral-digits*E*exponential-digits*; additional flags are needed to successfully parse strings in exponential notation with such elements as positive or negative signs and decimal point symbols.
AllowLeadingWhite	The *ws* element at the beginning of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the beginning of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.

NumberStyles value	Elements permitted in s in addition to digits
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The , element.
AllowCurrencySymbol	The *$* element.
Currency	All. The `s` parameter cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the beginning or end of `s`, *sign* at the beginning of `s`, and the . symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, thousands separator (*,*), and decimal point (.) elements.
Any	All styles, except `s` cannot represent a hexadecimal number.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific formatting information. When the TryParse(String, NumberStyles, IFormatProvider, Single) method is invoked, it calls the `provider` parameter's GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information about the format of the `s` parameter. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the formatting of `s` is interpreted based on the NumberFormatInfo object of the current culture.

If `s` is out of range of the Single data type, the method throws an OverflowException on .NET Framework and .NET Core 2.2 and earlier versions. On .NET Core 3.0 and

later versions, it returns Single.NegativeInfinity if `s` is less than Single.MinValue and Single.PositiveInfinity if `s` is greater than Single.MaxValue.

If a separator is encountered in the `s` parameter during a parse operation, and the applicable currency or number decimal and group separators are the same, the parse operation assumes that the separator is a decimal separator rather than a group separator. For more information about separators, see CurrencyDecimalSeparator, NumberDecimalSeparator, CurrencyGroupSeparator, and NumberGroupSeparator.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Equality(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Single values are equal.

```
C#
public static bool operator == (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are equal; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Remarks

The Equality method defines the equality operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.GreaterThan(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Single value is greater than another specified Single value.

```
C#
public static bool operator > (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Remarks

The GreaterThan method defines the operation of the greater-than operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.GreaterThanOrEqual(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Single value is greater than or equal to another specified Single value.

C#

```
public static bool operator >= (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Remarks

The GreaterThanOrEqual method defines the operation of the greater-than-or-equal operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.Inequality(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two specified Single values are not equal.

```
C#
public static bool operator != (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` and `right` are not equal; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Remarks

The Inequality method defines the inequality operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.LessThan(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Single value is less than another specified Single value.

```
C#
public static bool operator < (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Remarks

The LessThan method defines the operation of the less-than operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.LessThanOrEqual(Single, Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether a specified Single value is less than or equal to another specified Single value.

C#

```
public static bool operator <= (float left, float right);
```

Parameters

`left` Single

The first value to compare.

`right` Single

The second value to compare.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Remarks

The LessThanOrEqual method defines the operation of the less-than-or-equal operator for Single values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

This conversion is not supported. No value is returned.

Implements

ToChar(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.To Decimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.To Double(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

C#

```
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.To Int32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#

int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.To SByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToSByte(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.To Single(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

C#

```
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, unchanged.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this Single value.

provider IFormatProvider

An object that supplies information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt16(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt32(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the Single instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt64(Single) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.IAdditionOperators<Single,Single,Single>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static float IAdditionOperators<float,float,float>.operator + (float left,
float right);
```

Parameters

`left` Single

The value to which `right` is added.

`right` Single

The value that's added to `left`.

Returns

Single

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IAdditiveIdentity<Single,Single>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static float
System.Numerics.IAdditiveIdentity<System.Single,System.Single>.AdditiveIdentity { get; }
```

Property Value

Single

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IBinaryNumber<Single>.AllBits Set Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static float System.Numerics.IBinaryNumber<System.Single>.AllBitsSet { get;
}
```

Property Value

Single

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IBitwiseOperators<Single,Single,Single>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static float IBitwiseOperators<float,float,float>.operator & (float left,
float right);
```

Parameters

`left` Single

The value to `and` with `right`.

`right` Single

The value to `and` with `left`.

Returns

Single

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IBitwiseOperators<Single,Single,Single>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static float IBitwiseOperators<float,float,float>.operator | (float left,
float right);
```

Parameters

`left` Single

The value to `or` with `right`.

`right` Single

The value to `or` with `left`.

Returns

Single

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IBitwiseOperators<Single,Single,Single>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static float IBitwiseOperators<float,float,float>.operator ^ (float left,
float right);
```

Parameters

`left` Single

The value to xor with `right`.

`right` Single

The value to xor with `left`.

Returns

Single

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IBitwise Operators<Single,Single,Single>.Ones Complement Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static float IBitwiseOperators<float,float,float>.operator ~ (float value);
```

Parameters

`value` Single

The value for which to compute the ones-complement.

Returns

Single

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IDecrementOperators<Single>.Decrement(Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static float IDecrementOperators<float>.operator -- (float value);
```

Parameters

`value` Single

The value to decrement.

Returns

Single

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IDivisionOperators<Single,Single,Single>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static float IDivisionOperators<float,float,float>.operator / (float left,
float right);
```

Parameters

`left` Single

The value that `right` divides.

`right` Single

The value that divides `left`.

Returns

Single

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IFloatingPoint<Single>.GetExponentByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<float>.GetExponentByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteExponentLittleEndian(Span<Byte>, Int32).

Implements

GetExponentByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IFloatingPoint<Single>.GetExponentShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current exponent.

C#

```
int IFloatingPoint<float>.GetExponentShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current exponent.

Implements

GetExponentShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IFloatingPoint<Single>.GetSignificandBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the current significand.

C#

```
int IFloatingPoint<float>.GetSignificandBitLength ();
```

Returns

Int32

The length, in bits, of the current significand.

Implements

GetSignificandBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Single.IFloatingPoint<Single>.GetSignificandByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

C#

```
int IFloatingPoint<float>.GetSignificandByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteSignificandLittleEndian(Span<Byte>, Int32).

Implements

GetSignificandByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IFloatingPoint<Single>.TryWriteExponentBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<float>.TryWriteExponentBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IFloatingPoint<Single>.TryWriteExponentLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current exponent, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<float>.TryWriteExponentLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current exponent should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the exponent was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteExponentLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IFloatingPoint<Single>.TryWriteSignificandBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in big-endian format, to a given span.

C#

```
bool IFloatingPoint<float>.TryWriteSignificandBigEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IFloatingPoint<Single>.TryWriteSignificandLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current significand, in little-endian format, to a given span.

C#

```
bool IFloatingPoint<float>.TryWriteSignificandLittleEndian (Span<byte>
destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current significand should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the significand was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteSignificandLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IFloatingPoint Constants<Single>.E Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `e`.

C#

```
static float System.Numerics.IFloatingPointConstants<System.Single>.E { get;
}
```

Property Value

Single

Implements

E

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Constants<Single>.Pi Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `pi`.

C#

```
static float System.Numerics.IFloatingPointConstants<System.Single>.Pi {
get; }
```

Property Value

Single

Implements

Pi

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Constants<Single>.Tau Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the mathematical constant `tau`.

C#

```
static float System.Numerics.IFloatingPointConstants<System.Single>.Tau {
get; }
```

Property Value

Single

Implements

Tau

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Ieee754<Single>.Epsilon Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the smallest value such that can be added to `0` that does not result in `0`.

```
C#

static float System.Numerics.IFloatingPointIeee754<System.Single>.Epsilon {
get; }
```

Property Value

Single

Implements

Epsilon

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Ieee754<Single>.NaN Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents `NaN`.

C#

```
static float System.Numerics.IFloatingPointIeee754<System.Single>.NaN { get;
}
```

Property Value

Single

Implements

NaN

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Ieee754<Single>.NegativeInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents negative `infinity`.

C#

```
static float
System.Numerics.IFloatingPointIeee754<System.Single>.NegativeInfinity { get;
}
```

Property Value

Single

Implements

NegativeInfinity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IFloatingPoint Ieee754<Single>.NegativeZero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents negative `zero`.

C#

```
static float
System.Numerics.IFloatingPointIeee754<System.Single>.NegativeZero { get; }
```

Property Value

Single

Implements

NegativeZero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.IFloatingPoint Ieee754<Single>.PositiveInfinity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that represents positive `infinity`.

C#

```
static float
System.Numerics.IFloatingPointIeee754<System.Single>.PositiveInfinity { get;
}
```

Property Value

Single

Implements

PositiveInfinity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IIncrementOperators<Single>.Increment(Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static float IIncrementOperators<float>.operator ++ (float value);
```

Parameters

`value` Single

The value to increment.

Returns

Single

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IMinMaxValue<Single>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static float System.Numerics.IMinMaxValue<System.Single>.MaxValue { get; }
```

Property Value

Single

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IMinMaxValue<Single>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static float System.Numerics.IMinMaxValue<System.Single>.MinValue { get; }
```

Property Value

Single

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IModulusOperators<Single,Single,Single>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#
static float IModulusOperators<float,float,float>.operator % (float left,
float right);
```

Parameters

`left` Single

The value that `right` divides.

`right` Single

The value that divides `left`.

Returns

Single

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IMultiplicativeIdentity<Single,Single>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static float
System.Numerics.IMultiplicativeIdentity<System.Single,System.Single>.MultiplicativeIdentity { get; }
```

Property Value

Single

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.IMultiplyOperators<Single,Single,Single>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static float IMultiplyOperators<float,float,float>.operator * (float left,
float right);
```

Parameters

`left` Single

The value that `right` multiplies.

`right` Single

The value that multiplies `left`.

Returns

Single

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.IsCanonical(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<float>.IsCanonical (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.IsComplexNumber(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<float>.IsComplexNumber (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.IsImaginaryNumber(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

```
C#
static bool INumberBase<float>.IsImaginaryNumber (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.IsZero(Single) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<float>.IsZero (float value);
```

Parameters

`value` Single

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static float System.Numerics.INumberBase<System.Single>.One { get; }
```

Property Value

Single

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.INumberBase<Single>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.Single>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<float>.TryConvertFromChecked<TOther> (TOther value,
out float result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Single

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Single.INumberBase<Single>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<float>.TryConvertFromSaturating<TOther> (TOther
value, out float result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Single

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<float>.TryConvertFromTruncating<TOther> (TOther
value, out float result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` Single

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<float>.TryConvertToChecked<TOther> (float value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Single

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Single.INumberBase<Single>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<float>.TryConvertToSaturating<TOther> (float value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Single

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<float>.TryConvertToTruncating<TOther> (float value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` Single

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.INumberBase<Single>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
static float System.Numerics.INumberBase<System.Single>.Zero { get; }
```

Property Value

Single

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.ISignedNumber<Single>.NegativeOne Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `-1` for the type.

C#

```
static float System.Numerics.ISignedNumber<System.Single>.NegativeOne { get;
}
```

Property Value

Single

Implements

NegativeOne

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Single.ISubtractionOperators<Single,Single,Single>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static float ISubtractionOperators<float,float,float>.operator - (float
left, float right);
```

Parameters

`left` Single

The value from which `right` is subtracted.

`right` Single

The value that's subtracted from `left`.

Returns

Single

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IUnaryNegationOperators<Single,Single>.UnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static float IUnaryNegationOperators<float,float>.operator - (float value);
```

Parameters

`value` Single

The value for which to compute the unary negation.

Returns

Single

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Single.IUnaryPlusOperators<Single,Single>.UnaryPlus(Single) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static float IUnaryPlusOperators<float,float>.operator + (float value);
```

Parameters

`value` Single

The value for which to compute the unary plus.

Returns

Single

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T>.Enumerator Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an enumerator for the elements of a Span<T>.

```
C#
public ref struct Span<T>.Enumerator
```

Type Parameters

T

Inheritance Object → ValueType → Span<T>.Enumerator

Remarks

The C# foreach of the C# language and the For Each...Next construct in Visual Basic hides the complexity of enumerators. Instead of directly manipulating the enumerator, using `foreach` or `For Each...Next` is recommended.

Initially, the enumerator is positioned before the first element in the Span<T>. At this position, Current is undefined. You must call MoveNext to advance the enumerator to the first item in the Span<T> before reading the value of Current.

Current returns the same value until MoveNext is called. MoveNext sets Current to the next item in the Span<T>.

If MoveNext passes the end of the Span<T>, MoveNext returns `false`. When the enumerator is at this state, subsequent calls to MoveNext also return `false` and Current is undefined. You cannot set Current to the first item in the Span<T> again; you must create a new enumerator instance instead.

The enumerator does not have exclusive access to the Span<T>. In addition, the underlying data on which the span is based can also be modified. Therefore,

enumerating through a span is intrinsically not a thread-safe procedure. To guarantee thread safety during enumeration, you must implement your own synchronization. For example, the following code has a race condition. It does not ensure that the span will be enumerated before the `ClearContents` method executes. As a result, the underlying array is cleared during enumeration of the span:

C#

```
using System;
using System.Threading.Tasks;

class Program
{
    private static readonly byte[] _array = new byte[5];

    static void Main()
    {
        new Random(42).NextBytes(_array);
        Span<byte> span = _array;

        Task.Run( () => ClearContents() );

       EnumerateSpan(span);
    }

    public static void ClearContents()
    {
        Task.Delay(20).Wait();
        lock (_array)
        {
           Array.Clear(_array, 0, _array.Length);
        }
    }

    public static void EnumerateSpan(Span<byte> span)
    {
        foreach (byte element in span)
        {
            Console.WriteLine(element);
            Task.Delay(10).Wait();
        }
    }
}
// The example displays output like the following:
//     62
//     23
//     186
//     0
//     0
```

If you synchronize access to the array before enumerating the span, as the revised version of the `EnumerateSpan` method does in the following example, the `ClearContents` method doesn't modify underlying span data during enumeration. Note that the example locks the underlying array on which the span is based.

C#

```
public static void EnumerateSpan(Span<byte> span)
{
    lock (_array)
    {
        foreach (byte element in span)
        {
            Console.WriteLine(element);
            Task.Delay(10).Wait();
        }
    }
}
// The example displays the following output:
//    62
//    23
//    186
//    150
//    174
```

Unlike some other enumerator structures in .NET, the Span<T>.Enumerator:

- Does not implement the IEnumerator or IEnumerator<T> interface. This is because Span<T>.Enumerator is a ref struct.
- Does not include a `Reset` method, which can set the enumerator to its initial position before the first element in the span. (The IEnumerator.Reset() method must be implemented as part of the interface, but most implementors either throw an exception or provide no implementation.)

Properties

Expand table

Current	Gets a reference to the item at the current position of the enumerator.

Methods

Expand table

MoveNext()	Advances the enumerator to the next item of the Span<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Enumerator.Current Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a reference to the item at the current position of the enumerator.

```
C#
public ref T Current { get; }
```

Property Value

T

The element in the Span<T> at the current position of the enumerator.

Remarks

`Current` is undefined under either of the following conditions:

- Immediately after the enumerator is created, the enumerator is positioned before the first element in the span. MoveNext must be called to advance the enumerator to the first element of the span before reading the value of `Current`.
- The last call to MoveNext returned `false`, which indicates the end of the span.

`Current` returns the same value until MoveNext is called. MoveNext sets `Current` to the next item in the span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T>.Enumerator.MoveNext Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Advances the enumerator to the next item of the Span<T>.

```
C#
public bool MoveNext ();
```

Returns

Boolean

`true` if the enumerator successfully advanced to the next item; `false` if the end of the span has been passed.

Remarks

After an enumerator is created, it is positioned before the first element in the span, and the first call to `MoveNext` advances the enumerator to the first item in the span.

If `MoveNext` passes the end of the span, `MoveNext` returns `false`. When the enumerator is at this state, subsequent calls to `MoveNext` also return `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a type-safe and memory-safe representation of a contiguous region of arbitrary memory.

C#

```
[System.Runtime.InteropServices.Marshalling.NativeMarshalling(typeof(System.
Runtime.InteropServices.Marshalling.SpanMarshaller<,>))]
public readonly ref struct Span<T>
```

Type Parameters

T

The type of items in the Span<T>.

Inheritance Object → ValueType → Span<T>

Attributes NativeMarshallingAttribute

Remarks

For more information about this API, see Supplemental API remarks for Span<T>.

Constructors

Expand table

Span<T>(T)	Creates a new Span<T> of length 1 around the specified reference.
Span<T>(T[])	Creates a new Span<T> object over the entirety of a specified array.
Span<T>(T[], Int32, Int32)	Creates a new Span<T> object that includes a specified number of elements of an array starting at a specified index.

Span<T>(Void*, Int32)	Creates a new Span<T> object from a specified number of `T` elements starting at a specified memory address.

Properties

Expand table

Empty	Returns an empty Span<T> object.
IsEmpty	Returns a value that indicates whether the current Span<T> is empty.
Item[Int32]	Gets the element at the specified zero-based index.
Length	Returns the length of the current span.

Methods

Expand table

Clear()	Clears the contents of this Span<T> object.
CopyTo(Span<T>)	Copies the contents of this Span<T> into a destination Span<T>.
Equals(Object)	**Obsolete.** Calls to this method are not supported.
Fill(T)	Fills the elements of this span with a specified value.
GetEnumerator()	Returns an enumerator for this Span<T>.
GetHashCode()	**Obsolete.** Throws a NotSupportedException.
GetPinnable Reference()	Returns a reference to an object of type T that can be used for pinning. This method is intended to support .NET compilers and is not intended to be called by user code.
Slice(Int32)	Forms a slice out of the current span that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current span starting at a specified index for a specified length.
ToArray()	Copies the contents of this span into a new array.

ToString()	Returns the string representation of this Span<T> object.
TryCopyTo(Span<T>)	Attempts to copy the current Span<T> to a destination Span<T> and returns a value that indicates whether the copy operation succeeded.

Operators

Expand table

Equality(Span<T>, Span<T>)	Returns a value that indicates whether two Span<T> objects are equal.
Implicit(ArraySegment<T> to Span<T>)	Defines an implicit conversion of an ArraySegment<T> to a Span<T>.
Implicit(Span<T> to ReadOnlySpan<T>)	Defines an implicit conversion of a Span<T> to a ReadOnlySpan<T>.
Implicit(T[] to Span<T>)	Defines an implicit conversion of an array to a Span<T>.
Inequality(Span<T>, Span<T>)	Returns a value that indicates whether two Span<T> objects are not equal.

Extension Methods

Expand table

ToImmutableArray<T>(Span<T>)	Converts the span to an immutable array.
BinarySearch<T>(Span<T>, IComparable<T>)	Searches an entire sorted Span<T> for a value using the specified IComparable<T> generic interface.
BinarySearch<T,TComparer>(Span<T>, T, TComparer)	Searches an entire sorted Span<T> for a specified value using the specified `TComparer` generic type.
BinarySearch<T,TComparable>(Span<T>, TComparable)	Searches an entire sorted Span<T> for a value using the specified `TComparable` generic type.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>)	Finds the length of any common prefix shared between `span` and `other`.
CommonPrefixLength<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Finds the length of any common prefix shared between `span` and `other`.
Contains<T>(Span<T>, T)	Indicates whether a specified value is found in a span. Values are compared using

	IEquatable{T}.Equals(T).
ContainsAny<T>(Span<T>, T, T)	Searches for an occurrence of `value0` or `value1`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, T, T, T)	Searches for an occurrence of `value0`, `value1`, or `value2`, and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, SearchValues<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for an occurrence of any of the specified `values` and returns `true` if found. If not found, returns `false`.
ContainsAnyExcept<T>(Span<T>, T)	Searches for any value other than the specified `value`.
ContainsAnyExcept<T>(Span<T>, T, T)	Searches for any value other than `value0` or `value1`.
ContainsAnyExcept<T>(Span<T>, T, T, T)	Searches for any value other than `value0`, `value1`, or `value2`.
ContainsAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for any value other than the specified `values`.
ContainsAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for any value other than the specified `values`.
ContainsAnyExceptInRange<T>(Span<T>, T, T)	Searches for any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
ContainsAnyInRange<T>(Span<T>, T, T)	Searches for any value in the range between `lowInclusive` and `highInclusive`, inclusive, and returns `true` if found. If not found, returns `false`.
Count<T>(Span<T>, T)	Counts the number of times the specified `value` occurs in the `span`.
Count<T>(Span<T>, ReadOnlySpan<T>)	Counts the number of times the specified `value` occurs in the `span`.
EndsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether the specified sequence appears at the end of a span.

IndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its first occurrence. Values are compared using IEquatable{T}.Equals(T).
IndexOfAny<T>(Span<T>, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, T, T, T)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the first index of any of the specified values.
IndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any of the specified values similar to calling IndexOf several times with the logical OR operator.
IndexOfAnyExcept<T>(Span<T>, T)	Searches for the first index of any value other than the specified `value`.
IndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the first index of any value other than the specified `value0` or `value1`.
IndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the first index of any value other than the specified `value0`, `value1`, or `value2`.
IndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the first index of any value other than the specified `values`.
IndexOfAnyExceptInRange<T>(Span<T>, T, T)	Searches for the first index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
IndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the first index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOf<T>(Span<T>, T)	Searches for the specified value and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).

LastIndexOf<T>(Span<T>, ReadOnlySpan<T>)	Searches for the specified sequence and returns the index of its last occurrence. Values are compared using IEquatable{T}.Equals(T).
LastIndexOfAny<T>(Span<T>, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, T, T, T)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAny<T>(Span<T>, SearchValues<T>)	Searches for the last index of any of the specified values.
LastIndexOfAny<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any of the specified values similar to calling LastIndexOf several times with the logical OR operator.
LastIndexOfAnyExcept<T>(Span<T>, T)	Searches for the last index of any value other than the specified `value`.
LastIndexOfAnyExcept<T>(Span<T>, T, T)	Searches for the last index of any value other than the specified `value0` or `value1`.
LastIndexOfAnyExcept<T>(Span<T>, T, T, T)	Searches for the last index of any value other than the specified `value0`, `value1`, or `value2`.
LastIndexOfAnyExcept<T>(Span<T>, SearchValues<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExcept<T>(Span<T>, ReadOnlySpan<T>)	Searches for the last index of any value other than the specified `values`.
LastIndexOfAnyExceptInRange<T>(Span<T>, T, T)	Searches for the last index of any value outside of the range between `lowInclusive` and `highInclusive`, inclusive.
LastIndexOfAnyInRange<T>(Span<T>, T, T)	Searches for the last index of any value in the range between `lowInclusive` and `highInclusive`, inclusive.
Overlaps<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span overlap in memory.
Overlaps<T>(Span<T>, ReadOnlySpan<T>, Int32)	Determines whether a span and a read-only span overlap in memory and outputs the element offset.
Replace<T>(Span<T>, T, T)	Replaces all occurrences of `oldValue` with `newValue`.

Reverse<T>(Span<T>)	Reverses the sequence of the elements in the entire span.
SequenceCompareTo<T>(Span<T>, ReadOnlySpan<T>)	Determines the relative order of a span and a read-only span by comparing the elements using IComparable{T}.CompareTo(T).
SequenceEqual<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a span and a read-only span are equal by comparing the elements using IEquatable{T}.Equals(T).
SequenceEqual<T>(Span<T>, ReadOnlySpan<T>, IEqualityComparer<T>)	Determines whether two sequences are equal by comparing the elements using an IEqualityComparer<T>.
Sort<T>(Span<T>)	Sorts the elements in the entire Span<T> using the IComparable<T> implementation of each element of the Span<T>.
Sort<T>(Span<T>, Comparison<T>)	Sorts the elements in the entire Span<T> using the specified Comparison<T>.
Sort<T,TComparer>(Span<T>, TComparer)	Sorts the elements in the entire Span<T> using the `TComparer`.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the IComparable<T> implementation of each key.
Sort<TKey,TValue>(Span<TKey>, Span<TValue>, Comparison<TKey>)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparison.
Sort<TKey,TValue,TComparer>(Span<TKey>, Span<TValue>, TComparer)	Sorts a pair of spans (one containing the keys and the other containing the corresponding items) based on the keys in the first Span<T> using the specified comparer.
StartsWith<T>(Span<T>, ReadOnlySpan<T>)	Determines whether a specified sequence appears at the start of a span.
Trim<T>(Span<T>, T)	Removes all leading and trailing occurrences of a specified element from a span.
Trim<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading and trailing occurrences of a set of elements specified in a read-only span from a span.

TrimEnd<T>(Span<T>, T)	Removes all trailing occurrences of a specified element from a span.
TrimEnd<T>(Span<T>, ReadOnlySpan<T>)	Removes all trailing occurrences of a set of elements specified in a read-only span from a span.
TrimStart<T>(Span<T>, T)	Removes all leading occurrences of a specified element from the span.
TrimStart<T>(Span<T>, ReadOnlySpan<T>)	Removes all leading occurrences of a set of elements specified in a read-only span from the span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

See also

- Memory- and span-related types
- Memory<T> and Span<T> usage guidelines

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Span<T>(T)	Creates a new Span<T> of length 1 around the specified reference.
Span<T>(T[])	Creates a new Span<T> object over the entirety of a specified array.
Span<T>(Void*, Int32)	Creates a new Span<T> object from a specified number of `T` elements starting at a specified memory address.
Span<T>(T[], Int32, Int32)	Creates a new Span<T> object that includes a specified number of elements of an array starting at a specified index.

Span<T>(T)

Creates a new Span<T> of length 1 around the specified reference.

C#

```
public Span (ref T reference);
```

Parameters

`reference` T

A reference to data.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Span<T>(T[])

Creates a new Span<T> object over the entirety of a specified array.

```
C#
public Span (T[]? array);
```

Parameters

`array` T[]

The array from which to create the Span<T> object.

Exceptions

ArrayTypeMismatchException

`T` is a reference type, and `array` is not an array of type `T`.

Remarks

If `array` is `null`, this constructor returns a `null` `Span<T>`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Span<T>(Void*, Int32)

> Important

This API is not CLS-compliant.

Creates a new Span<T> object from a specified number of `T` elements starting at a specified memory address.

```
C#

[System.CLSCompliant(false)]
public Span (void* pointer, int length);
```

Parameters

`pointer` Void*

A pointer to the starting address of a specified number of `T` elements in memory.

`length` Int32

The number of `T` elements to be included in the Span<T>.

Attributes CLSCompliantAttribute

Exceptions

ArgumentException

`T` is a reference type or contains pointers and therefore cannot be stored in unmanaged memory.

ArgumentOutOfRangeException

`length` is negative.

Remarks

This constructor should be used with care, since it creates arbitrarily typed `T`s from a `void*`-typed block of memory, and `length` is not checked. However, if the constructor succeeds in creating a new Span<T> object, all subsequent uses are correct.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Span<T>(T[], Int32, Int32)

Creates a new Span<T> object that includes a specified number of elements of an array starting at a specified index.

```
C#

public Span (T[]? array, int start, int length);
```

Parameters

`array` T[]

The source array.

`start` Int32

The index of the first element to include in the new Span<T>.

`length` Int32

The number of elements to include in the new Span<T>.

Exceptions

ArgumentOutOfRangeException

`array` is `null`, but `start` or `length` is non-zero.

-or-

`start` is outside the bounds of the array.

-or-

`start` and `length` exceeds the number of elements in the array.

ArrayTypeMismatchException

`T` is a reference type, and `array` is not an array of type `T`.

Remarks

This method returns `default` when `array` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Empty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an empty Span<T> object.

C#

```
public static Span<T> Empty { get; }
```

Property Value

Span<T>

An empty Span<T> object.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.IsEmpty Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Span<T> is empty.

C#

```
public bool IsEmpty { get; }
```

Property Value

Boolean

`true` if the current span is empty; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the element at the specified zero-based index.

```
C#
public ref T this[int index] { get; }
```

Parameters

`index` Int32

The zero-based index of the element.

Property Value

T

The element at the specified index.

Exceptions

IndexOutOfRangeException

`index` is less than zero or greater than or equal to Length.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T>.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the length of the current span.

C#

```
public int Length { get; }
```

Property Value

Int32

The length of the current span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Clear Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clears the contents of this Span<T> object.

C#

```
public void Clear ();
```

Remarks

The `Clear` method sets the items in the Span<T> object to their default values. It does not remove items from the Span<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.CopyTo(Span<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this Span<T> into a destination Span<T>.

```
C#
public void CopyTo (Span<T> destination);
```

Parameters

`destination` Span<T>

The destination Span<T> object.

Exceptions

ArgumentException

`destination` is shorter than the source Span<T>.

Remarks

This method copies all of `source` to `destination` even if `source` and `destination` overlap.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Equals() on Span will always throw an exception. Use the equality operator instead.

Calls to this method are not supported.

C#

```
[System.Obsolete("Equals() on Span will always throw an exception. Use the equality operator instead.")]
public override bool Equals (object? obj);
```

Parameters

obj Object

Not supported.

Returns

Boolean

Calls to this method are not supported.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Calls to this method are not supported.

Remarks

Calls to the Equals method are not supported. Calls to the Equals methods produce either of two results:

- If `obj` is a Span<T>, the method call generates compiler error CS1503: "cannot convert from 'System.Span' to 'object'." This is because Span<T> is a ref struct that cannot be boxed and therefore cannot be converted to an Object.
- If the type of `obj` is not a Span<T>, the method call throws a NotSupportedException.

To compare two Span<T> objects for equality, use the Equality comparison operator.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Standard	*(2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Fill(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Fills the elements of this span with a specified value.

C#

```
public void Fill (T value);
```

Parameters

`value` T

The value to assign to each element of the span.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator for this Span<T>.

C#

```
public Span<T>.Enumerator GetEnumerator ();
```

Returns

Span<T>.Enumerator

An enumerator for this span.

Remarks

Instead of calling the GetEnumerator method directly, you can use the C# `foreach` statement and the Visual Basic `For Each`...`Next` construct to enumerate a Span<T>.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Span<T>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> GetHashCode() on Span will always throw an exception.

Throws a NotSupportedException.

C#

```
[System.Obsolete("GetHashCode() on Span will always throw an exception.")]
public override int GetHashCode ();
```

Returns

Int32

Calls to this method always throw a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Calls to this method are not supported.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Standard	*(2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Span<T>.GetPinnableReference Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a reference to an object of type T that can be used for pinning.

This method is intended to support .NET compilers and is not intended to be called by user code.

C#

```
public ref T GetPinnableReference ();
```

Returns

T

A reference to the element of the span at index 0, or `null` if the span is empty.

Examples

The following example demonstrates creating an integer array, pinning it, and writing each element to the console.

C#

```
using System;

// Note: you must compile this sample using the unsafe flag.
// From the command line, type the following: csc sample.cs /unsafe

public class Example
{
    public static unsafe void Main()
    {
        int[] array = CreateInt32Array();

        // Create a span, pin it, and print its elements.
        Span<int> span = array.AsSpan();
        fixed (int* spanPtr = span)
```

```
        {
            Console.WriteLine($"Span contains {span.Length} elements:");
            for (int i = 0; i < span.Length; i++)
            {
                Console.WriteLine(spanPtr[i]);
            }
            Console.WriteLine();
        }

        // Create a read-only span, pin it, and print its elements.
        ReadOnlySpan<int> readonlyspan = array.AsSpan();
        fixed (int* readonlyspanPtr = readonlyspan)
        {
            Console.WriteLine($"ReadOnlySpan contains {readonlyspan.Length}
elements:");
            for (int i = 0; i < readonlyspan.Length; i++)
            {
                Console.WriteLine(readonlyspanPtr[i]);
            }
            Console.WriteLine();
        }
    }

    private static int[] CreateInt32Array()
    {
        return new int[] { 100, 200, 300, 400, 500 };
    }
}

// The example displays the following output:
//       Span contains 5 elements:
//       100
//       200
//       300
//       400
//       500
//
//       ReadOnlySpan contains 5 elements:
//       100
//       200
//       300
//       400
//       500
```

Remarks

Applications should not directly call `GetPinnableReference`. Instead, callers should use their language's normal pinning syntax, such as C#'s `fixed` statement.

If pinning a `Span<char>`, the resulting `char*` **is not** assumed to be null-terminated. This behavior is different from pinning a `string`, where the resulting `char*` is guaranteed to

be null-terminated.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Slice Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Slice(Int32)	Forms a slice out of the current span that begins at a specified index.
Slice(Int32, Int32)	Forms a slice out of the current span starting at a specified index for a specified length.

Slice(Int32)

Forms a slice out of the current span that begins at a specified index.

C#

```
public Span<T> Slice (int start);
```

Parameters

`start` Int32

The index at which to begin the slice.

Returns

Span<T>

A span that consists of all elements of the current span from `start` to the end of the span.

Exceptions

ArgumentOutOfRangeException

`start` is less than zero or greater than Length.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Slice(Int32, Int32)

Forms a slice out of the current span starting at a specified index for a specified length.

C#

```
public Span<T> Slice (int start, int length);
```

Parameters

`start` Int32

The index at which to begin this slice.

`length` Int32

The desired length for the slice.

Returns

Span<T>

A span that consists of `length` elements from the current span starting at `start`.

Exceptions

ArgumentOutOfRangeException

`start` or `start` + `length` is less than zero or greater than Length.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.ToArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this span into a new array.

C#

```
public T[] ToArray ();
```

Returns

T[]

An array containing the data in the current span.

Remarks

This method performs a heap allocation and therefore should be avoided if possible. Heap allocations are expected in APIs that work with arrays. Using such APIs is unavoidable if an alternative API overload that takes a Span<T> does not exist.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Span<T>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of this Span<T> object.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Span<T> object.

Remarks

For a `Span<Char>`, the `ToString` method returns a String that contains the characters pointed to by the Span<T>. Otherwise, it returns a String with the name of the type and the number of elements that the Span<T> contains.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Span<T>.TryCopyTo(Span<T>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Attempts to copy the current Span<T> to a destination Span<T> and returns a value that indicates whether the copy operation succeeded.

```
C#
public bool TryCopyTo (Span<T> destination);
```

Parameters

destination Span<T>

The target of the copy operation.

Returns

Boolean

`true` if the copy operation succeeded; otherwise, `false`.

Remarks

This method copies all of `source` to `destination` even if `source` and `destination` overlap.

If `destination` is shorter than the source Span<T>, this method returns `false`, and no data is written to `destination`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Equality(Span<T>, Span<T>) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two Span<T> objects are equal.

```
C#
public static bool operator == (Span<T> left, Span<T> right);
```

Parameters

`left` Span<T>

The first span to compare.

`right` Span<T>

The second span to compare.

Returns

Boolean

`true` if the two Span<T> objects are equal; otherwise, `false`.

Remarks

Two Span<T> objects are equal if they have the same length and the corresponding elements of `left` and `right` point to the same memory. Note that the test for equality does *not* attempt to determine whether the contents are equal.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(ArraySegment<T> to Span<T>)	Defines an implicit conversion of an ArraySegment<T> to a Span<T>.
Implicit(Span<T> to ReadOnlySpan<T>)	Defines an implicit conversion of a Span<T> to a ReadOnlySpan<T>.
Implicit(T[] to Span<T>)	Defines an implicit conversion of an array to a Span<T>.

Implicit(ArraySegment<T> to Span<T>)

Defines an implicit conversion of an ArraySegment<T> to a Span<T>.

C#

```
public static implicit operator Span<T> (ArraySegment<T> segment);
```

Parameters

`segment` ArraySegment<T>

The array segment to be converted to a Span<T>.

Returns

Span<T>

A span that corresponds to the array segment.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(Span<T> to ReadOnlySpan<T>)

Defines an implicit conversion of a Span<T> to a ReadOnlySpan<T>.

C#

```
public static implicit operator ReadOnlySpan<T> (Span<T> span);
```

Parameters

span Span<T>

The object to convert to a ReadOnlySpan<T>.

Returns

ReadOnlySpan<T>

A read-only span that corresponds to the current instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Implicit(T[] to Span<T>)

Defines an implicit conversion of an array to a Span<T>.

```
C#
public static implicit operator Span<T> (T[]? array);
```

Parameters

`array` T[]

The array to convert to a Span<T>.

Returns

Span<T>

The span that corresponds to `array`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Span<T>.Inequality(Span<T>, Span<T>) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether two Span<T> objects are not equal.

C#

```
public static bool operator != (Span<T> left, Span<T> right);
```

Parameters

left Span<T>

The first span to compare.

right Span<T>

The second span to compare.

Returns

Boolean

`true` if the two Span<T> objects are not equal; otherwise, `false`.

Remarks

Two Span<T> objects are not equal if they have different lengths or if the corresponding elements of `left` and `right` do not point to the same memory.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

StackOverflowException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the execution stack exceeds the stack size. This class cannot be inherited.

C#

```
public sealed class StackOverflowException : SystemException
```

Inheritance Object → Exception → SystemException → StackOverflowException

Examples

The following example uses a counter to ensure that the number of recursive calls to the `Execute` method do not exceed a maximum defined by the MAX_RECURSIVE_CALLS constant.

C#

```
using System;

public class Example
{
   private const int MAX_RECURSIVE_CALLS = 1000;
   static int ctr = 0;

   public static void Main()
   {
      Example ex = new Example();
      ex.Execute();
      Console.WriteLine("\nThe call counter: {0}", ctr);
   }

   private void Execute()
   {
      ctr++;
      if (ctr % 50 == 0)
         Console.WriteLine("Call number {0} to the Execute method", ctr);
```

```
        if (ctr <= MAX_RECURSIVE_CALLS)
           Execute();

        ctr--;
     }
  }
  // The example displays the following output:
  //       Call number 50 to the Execute method
  //       Call number 100 to the Execute method
  //       Call number 150 to the Execute method
  //       Call number 200 to the Execute method
  //       Call number 250 to the Execute method
  //       Call number 300 to the Execute method
  //       Call number 350 to the Execute method
  //       Call number 400 to the Execute method
  //       Call number 450 to the Execute method
  //       Call number 500 to the Execute method
  //       Call number 550 to the Execute method
  //       Call number 600 to the Execute method
  //       Call number 650 to the Execute method
  //       Call number 700 to the Execute method
  //       Call number 750 to the Execute method
  //       Call number 800 to the Execute method
  //       Call number 850 to the Execute method
  //       Call number 900 to the Execute method
  //       Call number 950 to the Execute method
  //       Call number 1000 to the Execute method
  //
  //       The call counter: 0
```

Remarks

`StackOverflowException` is thrown for execution stack overflow errors, typically in case of a very deep or unbounded recursion. So make sure your code doesn't have an infinite loop or infinite recursion.

`StackOverflowException` uses the HRESULT COR_E_STACKOVERFLOW, which has the value 0x800703E9. The Localloc intermediate language (IL) instruction throws `StackOverflowException`. For a list of initial property values for a `StackOverflowException` object, see the StackOverflowException constructors.

Starting with the .NET Framework 2.0, you can't catch a `StackOverflowException` object with a `try`/`catch` block, and the corresponding process is terminated by default. Consequently, you should write your code to detect and prevent a stack overflow. For example, if your app depends on recursion, use a counter or a state condition to terminate the recursive loop. See the Examples section for an illustration of this technique.

> **Note**
>
> Applying the **HandleProcessCorruptedStateExceptionsAttribute** attribute to a method that throws a `StackOverflowException` has no effect. You still cannot handle the exception from user code.

If your app hosts the common language runtime (CLR), it can specify that the CLR should unload the application domain where the stack overflow exception occurs and let the corresponding process continue. For more information, see ICLRPolicyManager Interface.

Constructors

Expand table

StackOverflowException()	Initializes a new instance of the StackOverflowException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The requested operation caused a stack overflow." This message takes into account the current system culture.
StackOverflowException(String)	Initializes a new instance of the StackOverflowException class with a specified error message.
StackOverflowException(String, Exception)	Initializes a new instance of the StackOverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)

Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StackOverflowException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the StackOverflowException class.

Overloads

Expand table

StackOverflowException()	Initializes a new instance of the StackOverflowException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The requested operation caused a stack overflow." This message takes into account the current system culture.
StackOverflowException(String)	Initializes a new instance of the StackOverflowException class with a specified error message.
StackOverflowException(String, Exception)	Initializes a new instance of the StackOverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

StackOverflowException()

Initializes a new instance of the StackOverflowException class, setting the Message property of the new instance to a system-supplied message that describes the error, such as "The requested operation caused a stack overflow." This message takes into account the current system culture.

C#

```
public StackOverflowException ();
```

Remarks

The following table shows the initial property values for an instance of StackOverflowException.

Expand table

Property	Value
InnerException	`null`.
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

StackOverflowException(String)

Initializes a new instance of the StackOverflowException class with a specified error message.

C#

```
public StackOverflowException (string? message);
```

Parameters

`message` String

A String that describes the error. The content of message is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of StackOverflowException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

StackOverflowException(String, Exception)

Initializes a new instance of the StackOverflowException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public StackOverflowException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is

raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of StackOverflowException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

STAThreadAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that the COM threading model for an application is single-threaded apartment (STA).

C#

```
[System.AttributeUsage(System.AttributeTargets.Method)]
public sealed class STAThreadAttribute : Attribute
```

Inheritance Object → Attribute → STAThreadAttribute

Attributes AttributeUsageAttribute

Remarks

Apply this attribute to the entry point method (the `Main()` method in C# and Visual Basic). It has no effect on other methods. To set the apartment state of threads you start in your code, use the Thread.SetApartmentState or Thread.TrySetApartmentState method before starting the thread.

> **Note**
>
> For an overview of COM threading models, see **Understanding and Using COM Threading Models**.

COM threading models only apply to applications that use COM interop. The COM threading model can be set to single-threaded apartment or multithreaded apartment. The application thread is only initialized for COM interop if the thread actually makes a call to a COM component. If COM interop is not used, then the thread is not initialized, and the STAThreadAttribute attribute, if it is present, has no effect.

Starting with the .NET Framework version 2.0, the default threading model for COM interop depends on the language in which you are developing your application, as the following table shows.

Expand table

Language	COM apartment model
C#	Multithreaded apartment
C++	Multithreaded apartment
Visual Basic	Single-threaded apartment

To change these defaults, you use the STAThreadAttribute attribute to set the threading model for the application, or call the Thread.SetApartmentState or Thread.TrySetApartmentState method before starting the thread to set the threading model for a particular thread. In C++, you can also use the /CLRTHREADATTRIBUTE linker option to specify the apartment model.

ASP.NET applications should set the `ASPCompat` attribute of the @ Page directive to `true` to force the page to be serviced by the STA thread pool.

Here are some of the cases in which you'll want to use the STAThreadAttribute attribute to explicitly set the threading model to single-threaded apartment:

- You're developing a Windows Forms app. Windows Forms apps must be single-threaded if they communicate with Windows system components such as the Clipboard or Windows common dialog boxes, or if they use system features such as drag-and-drop functionality. The Windows Forms Application template for C# automatically adds the STAThreadAttribute attribute to C# projects. Because the single-threaded apartment model is the default for Visual Basic, there is no need for the attribute.

- You're developing a C# app that calls a Visual Basic library, which, in turn, relies on COM interop. Because the single-threaded apartment model is the default for Visual Basic, you should change your app's threading model to single-threaded by using the STAThreadAttribute attribute.

- Your application makes calls to COM components that use the single-threaded apartment model.

Constructors

Expand table

STAThreadAttribute()	Initializes a new instance of the STAThreadAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MTAThreadAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

STAThreadAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the STAThreadAttribute class.

C#

```
public STAThreadAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MTAThreadAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents text as a sequence of UTF-16 code units.

C#

```
public sealed class String : ICloneable, IComparable, IComparable<string>,
IConvertible, IEquatable<string>, IParsable<string>, ISpanParsable<string>,
System.Collections.Generic.IEnumerable<char>
```

Inheritance Object → String

Implements IEnumerable<Char> , IEnumerable , IComparable , IComparable<String> , IConvertible , IEquatable<String> , ICloneable , IParsable<String> , IParsable<TSelf> , ISpanParsable<String>

Remarks

For more information about this API, see Supplemental API remarks for String.

Constructors

Expand table

String(Char*)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters.
String(Char*, Int32, Int32)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters, a starting character position within that array, and a length.
String(Char, Int32)	Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times.

String(Char[])	Initializes a new instance of the String class to the Unicode characters indicated in the specified character array.
String(Char[], Int32, Int32)	Initializes a new instance of the String class to the value indicated by an array of Unicode characters, a starting character position within that array, and a length.
String(ReadOnlySpan<Char>)	Initializes a new instance of the String class to the Unicode characters indicated in the specified read-only span.
String(SByte*)	Initializes a new instance of the String class to the value indicated by a pointer to an array of 8-bit signed integers.
String(SByte*, Int32, Int32)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, and a length.
String(SByte*, Int32, Int32, Encoding)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, a length, and an Encoding object.

Fields

Expand table

Empty	Represents the empty string. This field is read-only.

Properties

Expand table

Chars[Int32]	Gets the Char object at a specified position in the current String object.
Length	Gets the number of characters in the current String object.

Methods

Expand table

Clone()	Returns a reference to this instance of String.
Compare(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects and returns an integer that indicates their relative

	position in the sort order.
Compare(String, Int32, String, Int32, Int32, Boolean)	Compares substrings of two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
Compare(String, Int32, String, Int32, Int32, Boolean, CultureInfo)	Compares substrings of two specified String objects, ignoring or honoring their case and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions)	Compares substrings of two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two substrings to each other in the sort order.
Compare(String, Int32, String, Int32, Int32, StringComparison)	Compares substrings of two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
Compare(String, String)	Compares two specified String objects and returns an integer that indicates their relative position in the sort order.
Compare(String, String, Boolean)	Compares two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
Compare(String, String, Boolean, Culture Info)	Compares two specified String objects, ignoring or honoring their case, and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
Compare(String, String, CultureInfo, CompareOptions)	Compares two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two strings to each other in the sort order.
Compare(String, String, String Comparison)	Compares two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
CompareOrdinal(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.
CompareOrdinal(String, String)	Compares two specified String objects by evaluating the numeric values of the corresponding Char objects

	in each string.
CompareTo(Object)	Compares this instance with a specified Object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.
CompareTo(String)	Compares this instance with a specified String object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
Concat(IEnumerable<String>)	Concatenates the members of a constructed IEnumerable<T> collection of type String.
Concat(Object)	Creates the string representation of a specified object.
Concat(Object, Object)	Concatenates the string representations of two specified objects.
Concat(Object, Object, Object)	Concatenates the string representations of three specified objects.
Concat(Object[])	Concatenates the string representations of the elements in a specified Object array.
Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of two specified read-only character spans.
Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of three specified read-only character spans.
Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of four specified read-only character spans.
Concat(String, String)	Concatenates two specified instances of String.
Concat(String, String, String)	Concatenates three specified instances of String.
Concat(String, String, String, String)	Concatenates four specified instances of String.
Concat(String[])	Concatenates the elements of a specified String array.
Concat<T>(IEnumerable<T>)	Concatenates the members of an IEnumerable<T> implementation.
Contains(Char)	Returns a value indicating whether a specified character occurs within this string.
Contains(Char, StringComparison)	Returns a value indicating whether a specified character occurs within this string, using the specified

	comparison rules.
Contains(String)	Returns a value indicating whether a specified substring occurs within this string.
Contains(String, StringComparison)	Returns a value indicating whether a specified string occurs within this string, using the specified comparison rules.
Copy(String)	**Obsolete.** Creates a new instance of String with the same value as a specified String.
CopyTo(Int32, Char[], Int32, Int32)	Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
CopyTo(Span<Char>)	Copies the contents of this string into the destination span.
Create(IFormatProvider, DefaultInterpolatedStringHandler)	Creates a new string by using the specified provider to control the formatting of the specified interpolated string.
Create(IFormatProvider, Span<Char>, DefaultInterpolatedStringHandler)	Creates a new string by using the specified provider to control the formatting of the specified interpolated string.
Create<TState>(Int32, TState, SpanAction<Char,TState>)	Creates a new string with a specific length and initializes it after creation by using the specified callback.
EndsWith(Char)	Determines whether the end of this string instance matches the specified character.
EndsWith(String)	Determines whether the end of this string instance matches the specified string.
EndsWith(String, Boolean, CultureInfo)	Determines whether the end of this string instance matches the specified string when compared using the specified culture.
EndsWith(String, StringComparison)	Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.
EnumerateRunes()	Returns an enumeration of Rune from this string.
Equals(Object)	Determines whether this instance and a specified object, which must also be a String object, have the same value.

Equals(String)	Determines whether this instance and another specified String object have the same value.
Equals(String, String)	Determines whether two specified String objects have the same value.
Equals(String, String, StringComparison)	Determines whether two specified String objects have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.
Equals(String, StringComparison)	Determines whether this string and a specified String object have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.
Format(IFormatProvider, CompositeFormat, Object[])	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format(IFormatProvider, CompositeFormat, ReadOnlySpan<Object>)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format(IFormatProvider, String, Object)	Replaces the format item or items in a specified string with the string representation of the corresponding object. A parameter supplies culture-specific formatting information.
Format(IFormatProvider, String, Object, Object)	Replaces the format items in a string with the string representation of two specified objects. A parameter supplies culture-specific formatting information.
Format(IFormatProvider, String, Object, Object, Object)	Replaces the format items in a string with the string representation of three specified objects. An parameter supplies culture-specific formatting information.
Format(IFormatProvider, String, Object[])	Replaces the format items in a string with the string representations of corresponding objects in a specified array. A parameter supplies culture-specific formatting information.
Format(String, Object)	Replaces one or more format items in a string with the string representation of a specified object.
Format(String, Object, Object)	Replaces the format items in a string with the string representation of two specified objects.
Format(String, Object, Object, Object)	Replaces the format items in a string with the string representation of three specified objects.
Format(String, Object[])	Replaces the format item in a specified string with the string representation of a corresponding object in a

	specified array.
Format<TArg0,TArg1,TArg2>(IFormatProvider, CompositeFormat, TArg0, TArg1, TArg2)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format<TArg0,TArg1>(IFormatProvider, CompositeFormat, TArg0, TArg1)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format<TArg0>(IFormatProvider, CompositeFormat, TArg0)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
GetEnumerator()	Retrieves an object that can iterate through the individual characters in this string.
GetHashCode()	Returns the hash code for this string.
GetHashCode(ReadOnlySpan<Char>)	Returns the hash code for the provided read-only character span.
GetHashCode(ReadOnlySpan<Char>, StringComparison)	Returns the hash code for the provided read-only character span using the specified rules.
GetHashCode(StringComparison)	Returns the hash code for this string using the specified rules.
GetPinnableReference()	Returns a reference to the element of the string at index zero. This method is intended to support .NET compilers and is not intended to be called by user code.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetTypeCode()	Returns the TypeCode for the String class.
IndexOf(Char)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string.
IndexOf(Char, Int32)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string. The search starts at a specified character position.
IndexOf(Char, Int32, Int32)	Reports the zero-based index of the first occurrence of the specified character in this instance. The search starts at a specified character position and examines a specified number of character positions.
IndexOf(Char, StringComparison)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string. A

	parameter specifies the type of search to use for the specified character.
IndexOf(String)	Reports the zero-based index of the first occurrence of the specified string in this instance.
IndexOf(String, Int32)	Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position.
IndexOf(String, Int32, Int32)	Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position and examines a specified number of character positions.
IndexOf(String, Int32, Int32, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string, the number of characters in the current string to search, and the type of search to use for the specified string.
IndexOf(String, Int32, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string and the type of search to use for the specified string.
IndexOf(String, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. A parameter specifies the type of search to use for the specified string.
IndexOfAny(Char[])	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters.
IndexOfAny(Char[], Int32)	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.
IndexOfAny(Char[], Int32, Int32)	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position and examines a specified number of character positions.
Insert(Int32, String)	Returns a new string in which a specified string is inserted at a specified index position in this instance.

Intern(String)	Retrieves the system's reference to the specified String.
IsInterned(String)	Retrieves a reference to a specified String.
IsNormalized()	Indicates whether this string is in Unicode normalization form C.
IsNormalized(NormalizationForm)	Indicates whether this string is in the specified Unicode normalization form.
IsNullOrEmpty(String)	Indicates whether the specified string is `null` or an empty string ("").
IsNullOrWhiteSpace(String)	Indicates whether a specified string is `null`, empty, or consists only of white-space characters.
Join(Char, Object[])	Concatenates the string representations of an array of objects, using the specified separator between each member.
Join(Char, String[])	Concatenates an array of strings, using the specified separator between each member.
Join(Char, String[], Int32, Int32)	Concatenates an array of strings, using the specified separator between each member, starting with the element in `value` located at the `startIndex` position, and concatenating up to `count` elements.
Join(String, IEnumerable<String>)	Concatenates the members of a constructed IEnumerable<T> collection of type String, using the specified separator between each member.
Join(String, Object[])	Concatenates the elements of an object array, using the specified separator between each element.
Join(String, String[])	Concatenates all the elements of a string array, using the specified separator between each element.
Join(String, String[], Int32, Int32)	Concatenates the specified elements of a string array, using the specified separator between each element.
Join<T>(Char, IEnumerable<T>)	Concatenates the members of a collection, using the specified separator between each member.
Join<T>(String, IEnumerable<T>)	Concatenates the members of a collection, using the specified separator between each member.
LastIndexOf(Char)	Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance.

LastIndexOf(Char, Int32)	Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.
LastIndexOf(Char, Int32, Int32)	Reports the zero-based index position of the last occurrence of the specified Unicode character in a substring within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.
LastIndexOf(String)	Reports the zero-based index position of the last occurrence of a specified string within this instance.
LastIndexOf(String, Int32)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.
LastIndexOf(String, Int32, Int32)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.
LastIndexOf(String, Int32, Int32, StringComparison)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for the specified number of character positions. A parameter specifies the type of comparison to perform when searching for the specified string.
LastIndexOf(String, Int32, StringComparison)	Reports the zero-based index of the last occurrence of a specified string within the current String object. The search starts at a specified character position and proceeds backward toward the beginning of the string. A parameter specifies the type of comparison to perform when searching for the specified string.
LastIndexOf(String, StringComparison)	Reports the zero-based index of the last occurrence of a specified string within the current String object. A parameter specifies the type of search to use for the specified string.
LastIndexOfAny(Char[])	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.

LastIndexOfAny(Char[], Int32)	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string.
LastIndexOfAny(Char[], Int32, Int32)	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Normalize()	Returns a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.
Normalize(NormalizationForm)	Returns a new string whose textual value is the same as this string, but whose binary representation is in the specified Unicode normalization form.
PadLeft(Int32)	Returns a new string that right-aligns the characters in this instance by padding them with spaces on the left, for a specified total length.
PadLeft(Int32, Char)	Returns a new string that right-aligns the characters in this instance by padding them on the left with a specified Unicode character, for a specified total length.
PadRight(Int32)	Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.
PadRight(Int32, Char)	Returns a new string that left-aligns the characters in this string by padding them on the right with a specified Unicode character, for a specified total length.
Remove(Int32)	Returns a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
Remove(Int32, Int32)	Returns a new string in which a specified number of characters in the current instance beginning at a specified position have been deleted.

Replace(Char, Char)	Returns a new string in which all occurrences of a specified Unicode character in this instance are replaced with another specified Unicode character.
Replace(String, String)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
Replace(String, String, Boolean, CultureInfo)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided culture and case sensitivity.
Replace(String, String, StringComparison)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided comparison type.
ReplaceLineEndings()	Replaces all newline sequences in the current string with NewLine.
ReplaceLineEndings(String)	Replaces all newline sequences in the current string with `replacementText`.
Split(Char, Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on a specified delimiting character and, optionally, options. Splits a string into a maximum number of substrings based on the provided character separator, optionally omitting empty substrings from the result.
Split(Char, StringSplitOptions)	Splits a string into substrings based on a specified delimiting character and, optionally, options.
Split(Char[])	Splits a string into substrings based on specified delimiting characters.
Split(Char[], Int32)	Splits a string into a maximum number of substrings based on specified delimiting characters.
Split(Char[], Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on specified delimiting characters and, optionally, options.
Split(Char[], StringSplitOptions)	Splits a string into substrings based on specified delimiting characters and options.
Split(String, Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on a specified delimiting string and, optionally, options.

Split(String, StringSplitOptions)	Splits a string into substrings that are based on the provided string separator.
Split(String[], Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on specified delimiting strings and, optionally, options.
Split(String[], StringSplitOptions)	Splits a string into substrings based on a specified delimiting string and, optionally, options.
StartsWith(Char)	Determines whether this string instance starts with the specified character.
StartsWith(String)	Determines whether the beginning of this string instance matches the specified string.
StartsWith(String, Boolean, CultureInfo)	Determines whether the beginning of this string instance matches the specified string when compared using the specified culture.
StartsWith(String, StringComparison)	Determines whether the beginning of this string instance matches the specified string when compared using the specified comparison option.
Substring(Int32)	Retrieves a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
Substring(Int32, Int32)	Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.
ToCharArray()	Copies the characters in this instance to a Unicode character array.
ToCharArray(Int32, Int32)	Copies the characters in a specified substring in this instance to a Unicode character array.
ToLower()	Returns a copy of this string converted to lowercase.
ToLower(CultureInfo)	Returns a copy of this string converted to lowercase, using the casing rules of the specified culture.
ToLowerInvariant()	Returns a copy of this String object converted to lowercase using the casing rules of the invariant culture.
ToString()	Returns this instance of String; no actual conversion is performed.
ToString(IFormatProvider)	Returns this instance of String; no actual conversion is performed.

ToUpper()	Returns a copy of this string converted to uppercase.
ToUpper(CultureInfo)	Returns a copy of this string converted to uppercase, using the casing rules of the specified culture.
ToUpperInvariant()	Returns a copy of this String object converted to uppercase using the casing rules of the invariant culture.
Trim()	Removes all leading and trailing white-space characters from the current string.
Trim(Char)	Removes all leading and trailing instances of a character from the current string.
Trim(Char[])	Removes all leading and trailing occurrences of a set of characters specified in an array from the current string.
TrimEnd()	Removes all the trailing white-space characters from the current string.
TrimEnd(Char)	Removes all the trailing occurrences of a character from the current string.
TrimEnd(Char[])	Removes all the trailing occurrences of a set of characters specified in an array from the current string.
TrimStart()	Removes all the leading white-space characters from the current string.
TrimStart(Char)	Removes all the leading occurrences of a specified character from the current string.
TrimStart(Char[])	Removes all the leading occurrences of a set of characters specified in an array from the current string.
TryCopyTo(Span<Char>)	Copies the contents of this string into the destination span.

Operators

Expand table

Equality(String, String)	Determines whether two specified strings have the same value.
Implicit(String to ReadOnlySpan<Char>)	Defines an implicit conversion of a given string to a read-only span of characters.

Inequality(String, String)	Determines whether two specified strings have different values.

Explicit Interface Implementations

Expand table

IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	For a description of this member, see ToDateTime(IFormatProvider).
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).

IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the current String object.
IEnumerable<Char>.GetEnumerator()	Returns an enumerator that iterates through the current String object.
IParsable<String>.Parse(String, IFormatProvider)	Parses a string into a value.
IParsable<String>.TryParse(String, IFormatProvider, String)	
ISpanParsable<String>.Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
ISpanParsable<String>.TryParse(ReadOnlySpan<Char>, IFormatProvider, String)	

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some

	transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.

ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.

Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence

	or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value

	from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.

Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.

Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the

	maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the

	minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.

Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an

	IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in

	the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.

ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each

	element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
AsMemory(String)	Creates a new `ReadOnlyMemory<Char>` over the portion of the target string.
AsMemory(String, Index)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified index.
AsMemory(String, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string starting at a specified character position.
AsMemory(String, Int32, Int32)	Creates a new `ReadOnlyMemory<Char>` over a portion of the target string beginning at a specified position with a length.

AsMemory(String, Range)	Creates a new `ReadOnlyMemory<Char>` over a specified range of the target string.
AsSpan(String)	Creates a new read-only span over a string.
AsSpan(String, Index)	Creates a new ReadOnlySpan<T> over a portion of the target string from a specified position to the end of the string.
AsSpan(String, Int32)	Creates a new read-only span over a portion of the target string from a specified position to the end of the string.
AsSpan(String, Int32, Int32)	Creates a new read-only span over a portion of the target string from a specified position for a specified number of characters.
AsSpan(String, Range)	Creates a new ReadOnlySpan<T> over a portion of a target string using the range start and end indexes.
IsNormalized(String)	Indicates whether the specified string is in Unicode normalization form C.
IsNormalized(String, NormalizationForm)	Indicates whether a string is in a specified Unicode normalization form.
Normalize(String)	Normalizes a string to a Unicode normalization form C.
Normalize(String, NormalizationForm)	Normalizes a string to the specified Unicode normalization form.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the

	ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every

	document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

This type is thread safe.

See also

- IComparable
- ICloneable
- IConvertible
- IEnumerable
- StringBuilder
- CultureInfo
- Sorting Weight Tables for Windows (.NET Framework and .NET Core only)
- Default Unicode Collation Element Tables
- Formatting Types in .NET
- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

String Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the String class.

Overloads

Expand table

String(Char*)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters.
String(Char[])	Initializes a new instance of the String class to the Unicode characters indicated in the specified character array.
String(ReadOnlySpan<Char>)	Initializes a new instance of the String class to the Unicode characters indicated in the specified read-only span.
String(SByte*)	Initializes a new instance of the String class to the value indicated by a pointer to an array of 8-bit signed integers.
String(Char, Int32)	Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times.
String(Char*, Int32, Int32)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters, a starting character position within that array, and a length.
String(Char[], Int32, Int32)	Initializes a new instance of the String class to the value indicated by an array of Unicode characters, a starting character position within that array, and a length.
String(SByte*, Int32, Int32)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, and a length.

String(SByte*, Int32, Int32, Encoding)	Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, a length, and an Encoding object.

Remarks

For more information about this API, see Supplemental API remarks for String constructor.

String(Char*)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters.

C#

```
[System.CLSCompliant(false)]
public String (char* value);
```

Parameters

`value` Char*

A pointer to a null-terminated array of Unicode characters.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

The current process does not have read access to all the addressed characters.

ArgumentException

`value` specifies an array that contains an invalid Unicode character, or `value` specifies an address less than 64000.

Remarks

> ⓘ **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

String(Char[])

Initializes a new instance of the String class to the Unicode characters indicated in the specified character array.

C#

```
public String (char[]? value);
```

Parameters

`value` Char[]

An array of Unicode characters.

Remarks

> ⓘ **Note**

For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

String(ReadOnlySpan<Char>)

Initializes a new instance of the String class to the Unicode characters indicated in the specified read-only span.

C#

```
public String (ReadOnlySpan<char> value);
```

Parameters

`value` ReadOnlySpan<Char>

A read-only span of Unicode characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

String(SByte*)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the String class to the value indicated by a pointer to an array of 8-bit signed integers.

```
C#

[System.CLSCompliant(false)]
public String (sbyte* value);
```

Parameters

`value` SByte*

A pointer to a null-terminated array of 8-bit signed integers. The integers are interpreted using the current system code page encoding (that is, the encoding specified by Default).

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

A new instance of String could not be initialized using `value`, assuming `value` is encoded in ANSI.

ArgumentOutOfRangeException

The length of the new string to initialize, which is determined by the null termination character of `value`, is too large to allocate.

AccessViolationException

`value` specifies an invalid address.

Remarks

> ⓘ **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

String(Char, Int32)

Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times.

C#

```
public String (char c, int count);
```

Parameters

c Char

A Unicode character.

count Int32

The number of times `c` occurs.

Exceptions

ArgumentOutOfRangeException

`count` is less than zero.

Remarks

> ⓘ **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

String(Char*, Int32, Int32)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters, a starting character position within that array, and a length.

C#

```
[System.CLSCompliant(false)]
public String (char* value, int startIndex, int length);
```

Parameters

`value` Char*

A pointer to an array of Unicode characters.

`startIndex` Int32

The starting position within `value`.

`length` Int32

The number of characters within `value` to use.

Attributes CLSCompliantAttribute

Exceptions

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero, `value` + `startIndex` cause a pointer overflow, or the current process does not have read access to all the addressed characters.

ArgumentException

`value` specifies an array that contains an invalid Unicode character, or `value` + `startIndex` specifies an address less than 64000.

Remarks

> **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

String(Char[], Int32, Int32)

Initializes a new instance of the String class to the value indicated by an array of Unicode characters, a starting character position within that array, and a length.

```
C#
public String (char[] value, int startIndex, int length);
```

Parameters

`value` Char[]

An array of Unicode characters.

`startIndex` Int32

The starting position within `value`.

`length` Int32

The number of characters within `value` to use.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero.

-or-

The sum of `startIndex` and `length` is greater than the number of elements in `value`.

Remarks

ⓘ Note

For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

String(SByte*, Int32, Int32)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, and a length.

C#

```
[System.CLSCompliant(false)]
public String (sbyte* value, int startIndex, int length);
```

Parameters

`value` SByte*

A pointer to an array of 8-bit signed integers. The integers are interpreted using the current system code page encoding (that is, the encoding specified by Default).

`startIndex` Int32

The starting position within `value`.

`length` Int32

The number of characters within `value` to use.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero.

-or-

The address specified by `value` + `startIndex` is too large for the current platform; that is, the address calculation overflowed.

-or-

The length of the new string to initialize is too large to allocate.

ArgumentException

The address specified by `value` + `startIndex` is less than 64K.

-or-

A new instance of String could not be initialized using `value`, assuming `value` is encoded in ANSI.

AccessViolationException

`value`, `startIndex`, and `length` collectively specify an invalid address.

Remarks

> **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

String(SByte*, Int32, Int32, Encoding)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, a length, and an Encoding object.

C#

```
[System.CLSCompliant(false)]
public String (sbyte* value, int startIndex, int length,
System.Text.Encoding enc);
```

Parameters

`value` SByte*

A pointer to an array of 8-bit signed integers.

`startIndex` Int32

The starting position within `value`.

`length` Int32

The number of characters within `value` to use.

`enc` Encoding

An object that specifies how the array referenced by `value` is encoded. If `enc` is `null`, ANSI encoding is assumed.

Attributes CLSCompliantAttribute

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero.

-or-

The address specified by `value` + `startIndex` is too large for the current platform; that is, the address calculation overflowed.

-or-

The length of the new string to initialize is too large to allocate.

ArgumentException

The address specified by `value` + `startIndex` is less than 64K.

-or-

A new instance of String could not be initialized using `value`, assuming `value` is encoded as specified by `enc`.

AccessViolationException

`value`, `startIndex`, and `length` collectively specify an invalid address.

Remarks

> **Note**
>
> For examples and comprehensive usage information about this and other `String` constructor overloads, see the **String** constructor summary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Empty Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the empty string. This field is read-only.

C#

```
public static readonly string Empty;
```

Field Value

String

Remarks

The value of this field is the zero-length string, "".

In application code, this field is most commonly used in assignments to initialize a string variable to an empty string. To test whether the value of a string is either `null` or String.Empty, use the IsNullOrEmpty method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNullOrEmpty(String)
- IsNullOrWhiteSpace(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Chars[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Char object at a specified position in the current String object.

C#

```
public char this[int index] { get; }
```

Parameters

index Int32

A position in the current string.

Property Value

Char

The object at position `index`.

Exceptions

IndexOutOfRangeException

`index` is greater than or equal to the length of this object or less than zero.

Examples

The following example demonstrates how you can use this indexer in a routine to validate a string.

C#

```
Console.Write("Type a string : ");
string myString = Console.ReadLine();
for (int i = 0; i < myString.Length; i ++)
   if(Uri.IsHexDigit(myString[i]))
```

```
        Console.WriteLine("{0} is a hexadecimal digit.", myString[i]);
    else
        Console.WriteLine("{0} is not a hexadecimal digit.", myString[i]);
// The example produces output like the following:
//    Type a string : 3f5EaZ
//    3 is a hexadecimal digit.
//    f is a hexadecimal digit.
//    5 is a hexadecimal digit.
//    E is a hexadecimal digit.
//    a is a hexadecimal digit.
//    Z is not a hexadecimal digit.
```

Remarks

The `index` parameter is zero-based.

This property returns the Char object at the position specified by the `index` parameter. However, a Unicode character might be represented by more than one Char. Use the System.Globalization.StringInfo class to work with Unicode characters instead of Char objects. For more information, see the "Char Objects and Unicode Characters" section in the String class overview.

In C#, the Chars[] property is an indexer. In Visual Basic, it is the default property of the String class. Each Char object in the string can be accessed by using code such as the following.

C#

```
string str1 = "Test";
for (int ctr = 0; ctr <= str1.Length - 1; ctr++ )
   Console.Write("{0} ", str1[ctr]);
// The example displays the following output:
//      T e s t
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Char
- GetEnumerator()
- Int32

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of characters in the current String object.

C#

```
public int Length { get; }
```

Property Value

Int32

The number of characters in the current string.

Examples

The following example demonstrates the Length property.

C#

```
string str = "abcdefg";
Console.WriteLine("1) The length of '{0}' is {1}", str, str.Length);
Console.WriteLine("2) The length of '{0}' is {1}", "xyz", "xyz".Length);

int length = str.Length;
Console.WriteLine("3) The length of '{0}' is {1}", str, length);

// This example displays the following output:
//    1) The length of 'abcdefg' is 7
//    2) The length of 'xyz' is 3
//    3) The length of 'abcdefg' is 7
```

Remarks

The Length property returns the number of Char objects in this instance, not the number of Unicode characters. The reason is that a Unicode character might be represented by

more than one Char. Use the System.Globalization.StringInfo class to work with each Unicode character instead of each Char.

In some languages, such as C and C++, a null character indicates the end of a string. In .NET, a null character can be embedded in a string. When a string includes one or more null characters, they are included in the length of the total string. For example, in the following string, the substrings "abc" and "def" are separated by a null character. The Length property returns 7, which indicates that it includes the six alphabetic characters as well as the null character.

C#

```
string characters = "abc\u0000def";
Console.WriteLine(characters.Length); // Displays 7
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int32

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a reference to this instance of String.

C#

```
public object Clone ();
```

Returns

Object

This instance of String.

Implements

Clone()

Remarks

The return value is not an independent copy of this instance; it is simply another view of the same data. Use the Copy or CopyTo method to create a separate String object with the same value as this instance.

Because the Clone method simply returns the existing string instance, there is little reason to call it directly.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

See also

- Copy(String)
- CopyTo(Int32, Char[], Int32, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Compare Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two specified String objects and returns an integer that indicates their relative position in the sort order.

Overloads

Expand table

Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions)	Compares substrings of two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two substrings to each other in the sort order.
Compare(String, Int32, String, Int32, Int32, Boolean, CultureInfo)	Compares substrings of two specified String objects, ignoring or honoring their case and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
Compare(String, Int32, String, Int32, Int32, StringComparison)	Compares substrings of two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
Compare(String, Int32, String, Int32, Int32, Boolean)	Compares substrings of two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
Compare(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects and returns an integer that indicates their relative position in the sort order.
Compare(String, String)	Compares two specified String objects and returns an integer that indicates their relative position in the sort order.
Compare(String, String, Boolean, CultureInfo)	Compares two specified String objects, ignoring or honoring their case, and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.

Compare(String, String, StringComparison)	Compares two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
Compare(String, String, Boolean)	Compares two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
Compare(String, String, CultureInfo, CompareOptions)	Compares two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two strings to each other in the sort order.

Remarks

All overloads of the Compare method return a 32-bit signed integer indicating the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The first substring precedes the second substring in the sort order.
Zero	The substrings occur in the same position in the sort order, or `length` is zero.
Greater than zero	The first substring follows the second substring in the sort order.

> ⚠ **Warning**
>
> Whenever possible, you should call an overload of the **Compare** method that includes a **StringComparison** parameter. For more information, see **Best Practices for Using Strings**.

Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions)

Compares substrings of two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two substrings to each other in the sort order.

```C#
public static int Compare (string? strA, int indexA, string? strB, int indexB, int length, System.Globalization.CultureInfo? culture, System.Globalization.CompareOptions options);
```

Parameters

`strA` String

The first string to use in the comparison.

`indexA` Int32

The starting position of the substring within `strA`.

`strB` String

The second string to use in the comparison.

`indexB` Int32

The starting position of the substring within `strB`.

`length` Int32

The maximum number of characters in the substrings to compare.

`culture` CultureInfo

An object that supplies culture-specific comparison information. If `culture` is `null`, the current culture is used.

`options` CompareOptions

Options to use when performing the comparison (such as ignoring case or symbols).

Returns

Int32

An integer that indicates the lexical relationship between the two substrings, as shown in the following table.

Expand table

Value	Condition
Less than zero	The substring in `strA` precedes the substring in `strB` in the sort order.

Value	Condition
Zero	The substrings occur in the same position in the sort order, or `length` is zero.
Greater than zero	The substring in `strA` follows the substring in `strB` in the sort order.

Exceptions

ArgumentException

`options` is not a CompareOptions value.

ArgumentOutOfRangeException

`indexA` is greater than `strA.Length`.

-or-

`indexB` is greater than `strB.Length`.

-or-

`indexA`, `indexB`, or `length` is negative.

-or-

Either `strA` or `strB` is `null`, and `length` is greater than zero.

Examples

The following example uses the Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions) method to compare the last names of two people. It then lists them in alphabetical order.

C#

```
string name1 = "Jack Smith";
string name2 = "John Doe";

// Get position of character after the space character.
int index1 = name1.IndexOf(" ");
index1 = index1 < 0 ? 0 : ++index1;

int index2 = name2.IndexOf(" ");
index2 = index2 < 0 ? 0 : ++index2;

int length = Math.Max(name1.Length, name2.Length);
```

```
Console.WriteLine("Sorted alphabetically by last name:");
if (String.Compare(name1, index1, name2, index2, length,
                   new CultureInfo("en-US"), CompareOptions.IgnoreCase) <
0)
    Console.WriteLine("{0}\n{1}", name1, name2);
else
    Console.WriteLine("{0}\n{1}", name2, name1);

// The example displays the following output:
//       Sorted alphabetically by last name:
//       John Doe
//       Jack Smith
```

Remarks

The substrings to compare start in `strA` at position `indexA` and in `strB` at position `indexB`. The length of the first substring is the length of `strA` minus `indexA`. The length of the second substring is the length of `strB` minus `indexB`.

The number of characters to compare is the lesser of the lengths of the two substrings, and `length`. The `indexA`, `indexB`, and `length` parameters must be nonnegative.

The comparison uses the `culture` parameter to obtain culture-specific information, such as casing rules and the alphabetical order of individual characters. For example, a particular culture could specify that certain combinations of characters be treated as a single character, that uppercase and lowercase characters be compared in a particular way, or that the sort order of a character depends on the characters that precede or follow it.

> ⊗ **Caution**
>
> The **Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions)** method is designed primarily for use in sorting or alphabetizing operations. It should not be used when the primary purpose of the method call is to determine whether two substrings are equivalent (that is, when the purpose of the method call is to test for a return value of zero). To determine whether two strings are equivalent, call the **Equals** method.

One or both of `strA` and `strB` can be `null`. By definition, any string, including String.Empty, compares greater than a null reference, and two null references compare equal to each other.

The comparison can be further specified by the `options` parameter, which consists of one or more members of the System.Globalization.CompareOptions enumeration. However, because the purpose of this method is to conduct a culture-sensitive string comparison, the CompareOptions.Ordinal and CompareOptions.OrdinalIgnoreCase values have no effect.

The comparison terminates when an inequality is discovered or both substrings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, the string with the remaining characters is considered greater. The return value is the result of the last comparison performed.

Notes to Callers

Character sets include ignorable characters. The Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions) method does not consider these characters when it performs a linguistic or culture-sensitive comparison. To recognize ignorable characters in your comparison, supply a value of Ordinal or OrdinalIgnoreCase for the `options` parameter.

See also

- Equals

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Compare(String, Int32, String, Int32, Int32, Boolean, CultureInfo)

Compares substrings of two specified String objects, ignoring or honoring their case and using culture-specific information to influence the comparison, and returns an

integer that indicates their relative position in the sort order.

```C#
public static int Compare (string? strA, int indexA, string? strB, int indexB, int length, bool ignoreCase, System.Globalization.CultureInfo? culture);
```

Parameters

`strA` String

The first string to use in the comparison.

`indexA` Int32

The position of the substring within `strA`.

`strB` String

The second string to use in the comparison.

`indexB` Int32

The position of the substring within `strB`.

`length` Int32

The maximum number of characters in the substrings to compare.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

`culture` CultureInfo

An object that supplies culture-specific comparison information. If `culture` is `null`, the current culture is used.

Returns

Int32

An integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The substring in `strA` precedes the substring in `strB` in the sort order.
Zero	The substrings occur in the same position in the sort order, or `length` is zero.
Greater than zero	The substring in `strA` follows the substring in `strB` in the sort order.

Exceptions

ArgumentOutOfRangeException

`indexA` is greater than `strA`.Length.

-or-

`indexB` is greater than `strB`.Length.

-or-

`indexA`, `indexB`, or `length` is negative.

-or-

Either `strA` or `strB` is `null`, and `length` is greater than zero.

Examples

The following example compares two substrings using different cultures and ignoring the case of the substrings. The choice of culture affects how the letter "I" is compared.

C#

```
// Sample for String.Compare(String, Int32, String, Int32, Int32,
Boolean, CultureInfo)
using System;
using System.Globalization;

class Sample5
{
    public static void Main()
    {
        //                  0123456
        String str1 = "MACHINE";
        String str2 = "machine";
        String str;
```

```
        int result;

        Console.WriteLine();
        Console.WriteLine("str1 = '{0}', str2 = '{1}'", str1, str2);
        Console.WriteLine("Ignore case, Turkish culture:");
        result = String.Compare(str1, 4, str2, 4, 2, true, new
CultureInfo("tr-TR"));
        str = ((result < 0) ? "less than" : ((result > 0) ? "greater
than" : "equal to"));
        Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(4,
2), str1);
        Console.Write("{0} ", str);
        Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(4,
2), str2);

        Console.WriteLine();
        Console.WriteLine("Ignore case, invariant culture:");
        result = String.Compare(str1, 4, str2, 4, 2, true,
CultureInfo.InvariantCulture);
        str = ((result < 0) ? "less than" : ((result > 0) ? "greater
than" : "equal to"));
        Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(4,
2), str1);
        Console.Write("{0} ", str);
        Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(4,
2), str2);
    }
}
/*
This example produces the following results:

str1 = 'MACHINE', str2 = 'machine'
Ignore case, Turkish culture:
Substring 'IN' in 'MACHINE' is less than substring 'in' in 'machine'.

Ignore case, invariant culture:
Substring 'IN' in 'MACHINE' is equal to substring 'in' in 'machine'.
*/
```

Remarks

The substrings to compare start in `strA` at `indexA`, and in `strB` at `indexB`. Both `indexA` and `indexB` are zero-based; that is, the first character in `strA` and `strB` is at position zero, not position one. The length of the first substring is equal to the length of `strA` minus `indexA` plus one. The length of the second substring is equal to the length of `strB` minus `indexB` plus one.

The number of characters to compare is the lesser of the lengths of the two substrings, and `length`. The `indexA`, `indexB`, and `length` parameters must be

nonnegative.

The comparison uses the `culture` parameter to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

One or both comparands can be `null`. By definition, any string, including the empty string (""), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both substrings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, Int32, String, Int32, Int32, Boolean, CultureInfo) method does not consider these characters when it performs a linguistic or culture-sensitive comparison. To recognize ignorable characters in your comparison, call the Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions) method and supply a value of Ordinal or OrdinalIgnoreCase for the `options` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Compare(String, Int32, String, Int32, Int32, StringComparison)

Compares substrings of two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, int indexA, string? strB, int
indexB, int length, StringComparison comparisonType);
```

Parameters

`strA` String

The first string to use in the comparison.

`indexA` Int32

The position of the substring within `strA`.

`strB` String

The second string to use in the comparison.

`indexB` Int32

The position of the substring within `strB`.

`length` Int32

The maximum number of characters in the substrings to compare.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The substring in `strA` precedes the substring in `strB` in the sort order.
Zero	The substrings occur in the same position in the sort order, or the `length` parameter is zero.
Greater than zero	The substring in `strA` follows the substring in `strB` in the sort order.

Exceptions

ArgumentOutOfRangeException

`indexA` is greater than `strA`.Length.

-or-

`indexB` is greater than `strB`.Length.

-or-

`indexA`, `indexB`, or `length` is negative.

-or-

Either `indexA` or `indexB` is `null`, and `length` is greater than zero.

ArgumentException

`comparisonType` is not a StringComparison value.

Examples

The following example compares two substrings.

C#

```
String str1 = "machine";
String str2 = "device";
String str;
int result;

Console.WriteLine();
Console.WriteLine("str1 = '{0}', str2 = '{1}'", str1, str2);
result = String.Compare(str1, 2, str2, 0, 2);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" :
"equal to"));
Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(2, 2),
str1);
Console.Write("{0} ", str);
Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(0, 2),
str2);

/*
This example produces the following results:

str1 = 'machine', str2 = 'device'
Substring 'ch' in 'machine' is less than substring 'de' in 'device'.
*/
```

Remarks

The substrings to compare start in `strA` at `indexA` and in `strB` at `indexB`. Both `indexA` and `indexB` are zero-based; that is, the first character in `strA` and `strB` is at position zero, not position one. The length of the first substring is equal to the

length of `strA` minus `indexA` plus one. The length of the second substring is equal to the length of `strB` minus `indexB` plus one.

The number of characters to compare is the lesser of the lengths of the two substrings, and `length`. The `indexA`, `indexB`, and `length` parameters must be nonnegative.

The `comparisonType` parameter indicates whether the comparison should use the current or invariant culture, honor or ignore the case of the comparands, or use word (culture-sensitive) or ordinal (culture-insensitive) sort rules.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both substrings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, Int32, String, Int32, Int32, StringComparison) method does not consider these characters when it performs a linguistic or culture-sensitive comparison. To recognize ignorable characters in your comparison, supply a value of Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Compare(String, Int32, String, Int32, Int32, Boolean)

Compares substrings of two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, int indexA, string? strB, int
indexB, int length, bool ignoreCase);
```

Parameters

strA String

The first string to use in the comparison.

`indexA` Int32

The position of the substring within `strA`.

`strB` String

The second string to use in the comparison.

`indexB` Int32

The position of the substring within `strB`.

`length` Int32

The maximum number of characters in the substrings to compare.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The substring in `strA` precedes the substring in `strB` in the sort order.
Zero	The substrings occur in the same position in the sort order, or `length` is zero.
Greater than zero	The substring in `strA` follows the substring in `strB` in the sort order.

Exceptions

ArgumentOutOfRangeException

`indexA` is greater than `strA`.Length.

-or-

`indexB` is greater than `strB`.Length.

-or-

`indexA`, `indexB`, or `length` is negative.

-or-

Either `indexA` or `indexB` is `null`, and `length` is greater than zero.

Examples

The following example performs two comparisons of two substrings that only differ in case. The first comparison ignores case and the second comparison considers case.

C#

```
String str1 = "MACHINE";
String str2 = "machine";
String str;
int result;

Console.WriteLine();
Console.WriteLine("str1 = '{0}', str2 = '{1}'", str1, str2);

Console.WriteLine("Ignore case:");
result = String.Compare(str1, 2, str2, 2, 2, true);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" :
"equal to"));
Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(2, 2),
str1);
Console.Write("{0} ", str);
Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(2, 2),
str2);
Console.WriteLine();

Console.WriteLine("Honor case:");
result = String.Compare(str1, 2, str2, 2, 2, false);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" :
"equal to"));
Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(2, 2),
str1);
Console.Write("{0} ", str);
Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(2, 2),
str2);

/*
This example produces the following results:

str1 = 'MACHINE', str2 = 'machine'
```

```
Ignore case:
Substring 'CH' in 'MACHINE' is equal to substring 'ch' in 'machine'.

Honor case:
Substring 'CH' in 'MACHINE' is greater than substring 'ch' in 'machine'.
*/
```

Remarks

The substrings to compare start in `strA` at `indexA`, and in `strB` at `indexB`. Both `indexA` and `indexB` are zero-based; that is, the first character in `strA` and `strB` is at position zero. The length of the first substring is equal to the length of `strA` minus `indexA` plus one. The length of the second substring is equal to the length of `strB` minus `indexB` plus one.

The number of characters to compare is the lesser of the lengths of the two substrings, and `length`. The `indexA`, `indexB`, and `length` parameters must be nonnegative.

The comparison uses the current culture to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

> ⚠ **Warning**
>
> When comparing strings, you should call the **Compare(String, Int32, String, Int32, Int32, StringComparison)** method, which requires that you explicitly specify the type of string comparison that the method uses. For more information, see **Best Practices for Using Strings**.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both substrings have been compared. However, if the two strings compare equal to the end of one string,

and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

The path name needs to be compared in an invariant manner. The correct code to do this is as follows.

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, Int32, String, Int32, Int32, Boolean) method does not consider these characters when it performs a linguistic or culture-sensitive comparison. To recognize ignorable characters in your comparison, call the Compare(String, Int32, String, Int32, Int32, StringComparison) method and supply a value of Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Compare(String, Int32, String, Int32, Int32)

Compares substrings of two specified String objects and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, int indexA, string? strB, int
indexB, int length);
```

Parameters

`strA` String

The first string to use in the comparison.

`indexA` Int32

The position of the substring within `strA`.

`strB` String

The second string to use in the comparison.

`indexB` Int32

The position of the substring within `strB`.

`length` Int32

The maximum number of characters in the substrings to compare.

Returns

Int32

A 32-bit signed integer indicating the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The substring in `strA` precedes the substring in `strB` in the sort order.
Zero	The substrings occur in the same position in the sort order, or `length` is zero.
Greater than zero	The substring in `strA` follows the substring in `strB` in the sort order.

Exceptions

ArgumentOutOfRangeException

`indexA` is greater than `strA`.Length.

-or-

`indexB` is greater than `strB`.Length.

-or-

`indexA`, `indexB`, or `length` is negative.

-or-

Either `indexA` or `indexB` is `null`, and `length` is greater than zero.

Examples

The following example compares two substrings.

C#

```
String str1 = "machine";
String str2 = "device";
String str;
int result;

Console.WriteLine();
Console.WriteLine("str1 = '{0}', str2 = '{1}'", str1, str2);
```

```
result = String.Compare(str1, 2, str2, 0, 2);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" :
"equal to"));
Console.Write("Substring '{0}' in '{1}' is ", str1.Substring(2, 2),
str1);
Console.Write("{0} ", str);
Console.WriteLine("substring '{0}' in '{1}'.", str2.Substring(0, 2),
str2);

/*
This example produces the following results:

str1 = 'machine', str2 = 'device'
Substring 'ch' in 'machine' is less than substring 'de' in 'device'.
*/
```

Remarks

The substrings to compare start in `strA` at `indexA` and in `strB` at `indexB`. Both `indexA` and `indexB` are zero-based; that is, the first character in `strA` and `strB` is at position zero. The length of the first substring is equal to the length of `strA` minus `indexA` plus one. The length of the second substring is equal to the length of `strB` minus `indexB` plus one.

The number of characters to compare is the lesser of the lengths of the two substrings, and `length`. The `indexA`, `indexB`, and `length` parameters must be nonnegative.

The comparison uses the current culture to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

> ⚠ **Warning**
>
> When comparing strings, you should call the **Compare(String, Int32, String, Int32, Int32, StringComparison)** method, which requires that you explicitly specify the type of string comparison that the method uses. For more information, see **Best Practices for Using Strings**.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both substrings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, Int32, String, Int32, Int32) method does not consider these characters when it performs a linguistic or culture-sensitive comparison. To recognize ignorable characters in your comparison, call the Compare(String, Int32, String, Int32, Int32, StringComparison) method and supply a value of Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Compare(String, String)

Compares two specified String objects and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, string? strB);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	`strA` precedes `strB` in the sort order.
Zero	`strA` occurs in the same position as `strB` in the sort order.
Greater than zero	`strA` follows `strB` in the sort order.

Examples

The following example calls the Compare(String, String) method to compare three sets of strings.

C#

```
// Create upper-case characters from their Unicode code units.
String stringUpper = "\x0041\x0042\x0043";

// Create lower-case characters from their Unicode code units.
String stringLower = "\x0061\x0062\x0063";

// Display the strings.
Console.WriteLine("Comparing '{0}' and '{1}':",
                stringUpper, stringLower);

// Compare the uppercased strings; the result is true.
Console.WriteLine("The Strings are equal when capitalized? {0}",
                String.Compare(stringUpper.ToUpper(),
stringLower.ToUpper()) == 0
                               ? "true" : "false");

// The previous method call is equivalent to this Compare method, which
ignores case.
Console.WriteLine("The Strings are equal when case is ignored? {0}",
                String.Compare(stringUpper, stringLower, true) == 0
                               ? "true" : "false" );

// The example displays the following output:
//       Comparing 'ABC' and 'abc':
//       The Strings are equal when capitalized? true
//       The Strings are equal when case is ignored? true
```

In the following example, the `ReverseStringComparer` class demonstrates how you can evaluate two strings with the Compare method.

C#

```
using System;
using System.Text;
using System.Collections;

public class SamplesArrayList
{

    public static void Main()
    {
        // Creates and initializes a new ArrayList.
        ArrayList myAL = new ArrayList();
        myAL.Add("Eric");
        myAL.Add("Mark");
        myAL.Add("Lance");
        myAL.Add("Rob");
        myAL.Add("Kris");
        myAL.Add("Brad");
        myAL.Add("Kit");
        myAL.Add("Bradley");
        myAL.Add("Keith");
        myAL.Add("Susan");

        // Displays the properties and values of    the ArrayList.
        Console.WriteLine("Count: {0}", myAL.Count);

        PrintValues("Unsorted", myAL);
        myAL.Sort();
        PrintValues("Sorted", myAL);
        myAL.Sort(new ReverseStringComparer());
        PrintValues("Reverse", myAL);

        string[] names = (string[])myAL.ToArray(typeof(string));
    }
    public static void PrintValues(string title, IEnumerable myList)
    {
        Console.Write("{0,10}: ", title);
        StringBuilder sb = new StringBuilder();
        foreach (string s in myList)
        {
            sb.AppendFormat("{0}, ", s);
        }
        sb.Remove(sb.Length - 2, 2);
        Console.WriteLine(sb);
    }
}
public class ReverseStringComparer : IComparer
{
    public int Compare(object? x, object? y)
    {
        string? s1 = x as string;
        string? s2 = y as string;
        //negate the return value to get the reverse order
        return -String.Compare(s1, s2);
```

```
    }
}
```

Remarks

The comparison uses the current culture to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

> ⚠ **Warning**
>
> When comparing strings, you should call the **Compare(String, String, StringComparison)** method, which requires that you explicitly specify the type of string comparison that the method uses. For more information, see **Best Practices for Using Strings**.

One or both comparands can be `null`. By definition, any string, including the empty string (""), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both strings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
```

```
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

```C#
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, String) method does not consider such characters when it performs a culture-sensitive comparison. For example, if the following code is run on the .NET Framework 4 or later, a culture-sensitive comparison of "animal" with "ani-mal" (using a soft hyphen, or U+00AD) indicates that the two strings are equivalent.

```C#
string s1 = "ani\u00ADmal";
string s2 = "animal";

Console.WriteLine("Comparison of '{0}' and '{1}': {2}",
                s1, s2, String.Compare(s1, s2));

// The example displays the following output:
//       Comparison of 'ani-mal' and 'animal': 0
```

To recognize ignorable characters in a string comparison, call the Compare(String, String, StringComparison) method and supply a value of either Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Compare(String, String, Boolean, CultureInfo)

Compares two specified String objects, ignoring or honoring their case, and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, string? strB, bool ignoreCase,
System.Globalization.CultureInfo? culture);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

`culture` CultureInfo

An object that supplies culture-specific comparison information. If `culture` is `null`, the current culture is used.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	`strA` precedes `strB` in the sort order.
Zero	`strA` occurs in the same position as `strB` in the sort order.
Greater than zero	`strA` follows `strB` in the sort order.

Examples

The following example demonstrates how culture can affect a comparison. In Czech - Czech Republic culture, "ch" is a single character that is greater than "d". However, in English - United States culture, "ch" consists of two characters, and "c" is less than "d".

C#

```
public static void Main()
{
    String str1 = "change";
    String str2 = "dollar";
    String relation;

    relation = symbol(String.Compare(str1, str2, false, new
CultureInfo("en-US")));
    Console.WriteLine("For en-US: {0} {1} {2}", str1, relation, str2);

    relation = symbol(String.Compare(str1, str2, false, new
CultureInfo("cs-CZ")));
    Console.WriteLine("For cs-CZ: {0} {1} {2}", str1, relation, str2);
}

private static String symbol(int r)
{
    String s = "=";
    if (r < 0) s = "<";
    else if (r > 0) s = ">";
    return s;
}

/*
```

```
This example produces the following results.
For en-US: change < dollar
For cs-CZ: change > dollar
*/
```

Remarks

The comparison uses the `culture` parameter to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both strings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, String, Boolean, CultureInfo) method does not consider such characters when it performs a culture-sensitive comparison. For example, if the following code is run on the .NET Framework 4 or later, a case-insensitive comparison of "animal" with "Ani-mal" (using a soft hyphen, or U+00AD) using the invariant culture indicates that the two strings are equivalent.

C#

```
string s1 = "Ani\u00ADmal";
string s2 = "animal";

Console.WriteLine("Comparison of '{0}' and '{1}': {2}",
                s1, s2, String.Compare(s1, s2, true,
                CultureInfo.InvariantCulture));

// The example displays the following output:
//       Comparison of 'Ani-mal' and 'animal': 0
```

To recognize ignorable characters in a string comparison, call the Compare(String, String, CultureInfo, CompareOptions) method and supply a value of either Ordinal or OrdinalIgnoreCase for the `options` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)
- IsPrefix(String, String, CompareOptions)
- Boolean

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Compare(String, String, StringComparison)

Compares two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, string? strB, StringComparison
comparisonType);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	`strA` precedes `strB` in the sort order.

Value	Condition
Zero	`strA` is in the same position as `strB` in the sort order.
Greater than zero	`strA` follows `strB` in the sort order.

Exceptions

ArgumentException

`comparisonType` is not a StringComparison value.

NotSupportedException

StringComparison is not supported.

Examples

The following example compares three versions of the letter "I". The results are affected by the choice of culture, whether case is ignored, and whether an ordinal comparison is performed.

C#

```
// This example demonstrates the
// System.String.Compare(String, String, StringComparison) method.

using System;
using System.Threading;

class Sample
{
    public static void Main()
    {
        string intro = "Compare three versions of the letter I using
different " +
                       "values of StringComparison.";

        // Define an array of strings where each element contains a
version of the
        // letter I. (An array of strings is used so you can easily
modify this
        // code example to test additional or different combinations of
strings.)

        string[] threeIs = new string[3];
        // LATIN SMALL LETTER I (U+0069)
        threeIs[0] = "\u0069";
        // LATIN SMALL LETTER DOTLESS I (U+0131)
        threeIs[1] = "\u0131";
```

```
        // LATIN CAPITAL LETTER I (U+0049)
        threeIs[2] = "\u0049";

        string[] unicodeNames =
        {
            "LATIN SMALL LETTER I (U+0069)",
            "LATIN SMALL LETTER DOTLESS I (U+0131)",
            "LATIN CAPITAL LETTER I (U+0049)"
        };

        StringComparison[] scValues =
        {
            StringComparison.CurrentCulture,
            StringComparison.CurrentCultureIgnoreCase,
            StringComparison.InvariantCulture,
            StringComparison.InvariantCultureIgnoreCase,
            StringComparison.Ordinal,
            StringComparison.OrdinalIgnoreCase
        };

        Console.Clear();
        Console.WriteLine(intro);

        // Display the current culture because the culture-specific
comparisons
        // can produce different results with different cultures.
        Console.WriteLine(
            "The current culture is {0}.\n",
Thread.CurrentThread.CurrentCulture.Name);

        // Determine the relative sort order of three versions of the
letter I.
        foreach (StringComparison sc in scValues)
        {
            Console.WriteLine("StringComparison.{0}:", sc);

            // LATIN SMALL LETTER I (U+0069) : LATIN SMALL LETTER DOTLESS
I (U+0131)
            Test(0, 1, sc, threeIs, unicodeNames);

            // LATIN SMALL LETTER I (U+0069) : LATIN CAPITAL LETTER I
(U+0049)
            Test(0, 2, sc, threeIs, unicodeNames);

            // LATIN SMALL LETTER DOTLESS I (U+0131) : LATIN CAPITAL
LETTER I (U+0049)
            Test(1, 2, sc, threeIs, unicodeNames);

            Console.WriteLine();
        }
    }

    protected static void Test(
        int x, int y, StringComparison comparison, string[] testI,
string[] testNames)
```

```
    {
        string resultFmt = "{0} is {1} {2}";
        string result = "equal to";
        int cmpValue = 0;

        cmpValue = String.Compare(testI[x], testI[y], comparison);
        if (cmpValue < 0)
            result = "less than";
        else if (cmpValue > 0)
            result = "greater than";
        Console.WriteLine(resultFmt, testNames[x], result, testNames[y]);
    }
}

/*
This code example produces the following results:

Compare three versions of the letter I using different values of
StringComparison.
The current culture is en-US.

StringComparison.CurrentCulture:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is less than LATIN CAPITAL LETTER I
(U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

StringComparison.CurrentCultureIgnoreCase:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is equal to LATIN CAPITAL LETTER I (U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

StringComparison.InvariantCulture:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is less than LATIN CAPITAL LETTER I
(U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

StringComparison.InvariantCultureIgnoreCase:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is equal to LATIN CAPITAL LETTER I (U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

StringComparison.Ordinal:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is greater than LATIN CAPITAL LETTER I
```

```
(U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

StringComparison.OrdinalIgnoreCase:
LATIN SMALL LETTER I (U+0069) is less than LATIN SMALL LETTER DOTLESS I
(U+0131)
LATIN SMALL LETTER I (U+0069) is equal to LATIN CAPITAL LETTER I (U+0049)
LATIN SMALL LETTER DOTLESS I (U+0131) is greater than LATIN CAPITAL
LETTER I (U+0049)

*/
```

Remarks

The `comparisonType` parameter indicates whether the comparison should use the current or invariant culture, honor or ignore the case of the comparands, or use word (culture-sensitive) or ordinal (culture-insensitive) sort rules.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both strings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, String, StringComparison) method does not consider such characters when it performs a culture-sensitive comparison. To recognize ignorable characters in your comparison, supply a value of Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)
- CultureInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Compare(String, String, Boolean)

Compares two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.

C#

```
public static int Compare (string? strA, string? strB, bool ignoreCase);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	`strA` precedes `strB` in the sort order.
Zero	`strA` occurs in the same position as `strB` in the sort order.
Greater than zero	`strA` follows `strB` in the sort order.

Examples

The following example demonstrates that the Compare(String, String, Boolean) method is equivalent to using ToUpper or ToLower when comparing strings.

C#

```
// Create upper-case characters from their Unicode code units.
String stringUpper = "\x0041\x0042\x0043";

// Create lower-case characters from their Unicode code units.
String stringLower = "\x0061\x0062\x0063";

// Display the strings.
```

```
Console.WriteLine("Comparing '{0}' and '{1}':",
                stringUpper, stringLower);

// Compare the uppercased strings; the result is true.
Console.WriteLine("The Strings are equal when capitalized? {0}",
                String.Compare(stringUpper.ToUpper(),
stringLower.ToUpper()) == 0
                               ? "true" : "false");

// The previous method call is equivalent to this Compare method, which
ignores case.
Console.WriteLine("The Strings are equal when case is ignored? {0}",
                String.Compare(stringUpper, stringLower, true) == 0
                               ? "true" : "false" );

// The example displays the following output:
//       Comparing 'ABC' and 'abc':
//       The Strings are equal when capitalized? true
//       The Strings are equal when case is ignored? true
```

Remarks

The comparison uses the current culture to obtain culture-specific information such as casing rules and the alphabetic order of individual characters. For example, a culture could specify that certain combinations of characters be treated as a single character, or uppercase and lowercase characters be compared in a particular way, or that the sorting order of a character depends on the characters that precede or follow it.

The comparison is performed using word sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

> ⚠ **Warning**
>
> When comparing strings, you should call the **Compare(String, String, StringComparison)** method, which requires that you explicitly specify the type of string comparison that the method uses. For more information, see **Best Practices for Using Strings**.

One or both comparands can be `null`. By definition, any string, including the empty string ("), compares greater than a null reference; and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both strings have been compared. However, if the two strings compare equal to the end of one string,

and the other string has characters remaining, then the string with remaining characters is considered greater. The return value is the result of the last comparison performed.

Unexpected results can occur when comparisons are affected by culture-specific casing rules. For example, in Turkish, the following example yields the wrong results because the file system in Turkish does not use linguistic casing rules for the letter "i" in "file".

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5, true) == 0);
}
```

Compare the path name to "file" using an ordinal comparison. The correct code to do this is as follows:

C#

```
static bool IsFileURI(String path)
{
    return (String.Compare(path, 0, "file:", 0, 5,
StringComparison.OrdinalIgnoreCase) == 0);
}
```

Notes to Callers

Character sets include ignorable characters. The Compare(String, String, Boolean) method does not consider such characters when it performs a culture-sensitive comparison. For example, if the following code is run on the .NET Framework 4 or later, a culture-sensitive, case-insensitive comparison of "animal" with "Ani-mal" (using a soft hyphen, or U+00AD) indicates that the two strings are equivalent.

C#

```
string s1 = "Ani\u00ADmal";
string s2 = "animal";

Console.WriteLine("Comparison of '{0}' and '{1}': {2}",
                s1, s2, String.Compare(s1, s2, true));

// The example displays the following output:
//       Comparison of 'Ani-mal' and 'animal': 0
```

To recognize ignorable characters in a string comparison, call the Compare(String, String, StringComparison) method and supply a value of either Ordinal or OrdinalIgnoreCase for the `comparisonType` parameter.

See also

- Int32
- CompareOrdinal(String, String)
- CompareTo(Object)
- IsPrefix(String, String, CompareOptions)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Compare(String, String, CultureInfo, CompareOptions)

Compares two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two strings to each other in the sort order.

C#

```
public static int Compare (string? strA, string? strB,
System.Globalization.CultureInfo? culture,
System.Globalization.CompareOptions options);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

`culture` CultureInfo

The culture that supplies culture-specific comparison information. If `culture` is `null`, the current culture is used.

`options` CompareOptions

Options to use when performing the comparison (such as ignoring case or symbols).

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between `strA` and `strB`, as shown in the following table

Expand table

Value	Condition
Less than zero	`strA` precedes `strB` in the sort order.
Zero	`strA` occurs in the same position as `strB` in the sort order.
Greater than zero	`strA` follows `strB` in the sort order.

Exceptions

ArgumentException

`options` is not a CompareOptions value.

Examples

The following example compares two strings in three different ways: Use linguistic comparison for the en-US culture; using linguistic case-sensitive comparison for the en-US culture; and using an ordinal comparison. It illustrates how the three methods of comparison produce three different results.

C#

```
using System;
using System.Globalization;

public class Example0
{
    public static void Main()
    {
        string string1 = "brother";
        string string2 = "Brother";
        string relation;
        int result;

        // Cultural (linguistic) comparison.
        result = String.Compare(string1, string2, new CultureInfo("en-
US"),
                                CompareOptions.None);
        if (result > 0)
            relation = "comes after";
        else if (result == 0)
            relation = "is the same as";
        else
            relation = "comes before";

        Console.WriteLine("'{0}' {1} '{2}'.",
                          string1, relation, string2);

        // Cultural (linguistic) case-insensitive comparison.
        result = String.Compare(string1, string2, new CultureInfo("en-
US"),
                                CompareOptions.IgnoreCase);
        if (result > 0)
            relation = "comes after";
        else if (result == 0)
            relation = "is the same as";
        else
            relation = "comes before";

        Console.WriteLine("'{0}' {1} '{2}'.",
                          string1, relation, string2);

        // Culture-insensitive ordinal comparison.
        result = String.CompareOrdinal(string1, string2);
        if (result > 0)
            relation = "comes after";
        else if (result == 0)
            relation = "is the same as";
        else
            relation = "comes before";

        Console.WriteLine("'{0}' {1} '{2}'.",
                          string1, relation, string2);
```

```
        // The example produces the following output:
        //    'brother' comes before 'Brother'.
        //    'brother' is the same as 'Brother'.
        //    'brother' comes after 'Brother'.
    }
}
```

Remarks

The comparison uses the `culture` parameter to obtain culture-specific information, such as casing rules and the alphabetical order of individual characters. For example, a particular culture could specify that certain combinations of characters be treated as a single character, that uppercase and lowercase characters be compared in a particular way, or that the sort order of a character depends on the characters that precede or follow it.

> ⊗ **Caution**
>
> The **Compare(String, String, CultureInfo, CompareOptions)** method is designed primarily for use in sorting or alphabetizing operations. It should not be used when the primary purpose of the method call is to determine whether two strings are equivalent (that is, when the purpose of the method call is to test for a return value of zero). To determine whether two strings are equivalent, call the **Equals** method.

The comparison can be further specified by the `options` parameter, which consists of one or more members of the CompareOptions enumeration. However, because the purpose of this method is to conduct a culture-sensitive string comparison, the CompareOptions.Ordinal and CompareOptions.OrdinalIgnoreCase values have no effect.

Either or both comparands can be `null`. By definition, any string, including String.Empty, compares greater than a null reference, and two null references compare equal to each other.

The comparison terminates when an inequality is discovered or both strings have been compared. However, if the two strings compare equal to the end of one string, and the other string has characters remaining, the string with the remaining characters is considered greater.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. The Compare(String, String, CultureInfo, CompareOptions) method does not consider such characters when it performs a culture-sensitive comparison. To recognize ignorable characters in your comparison, supply a value of Ordinal or OrdinalIgnoreCase for the `options` parameter.

See also

- CompareOrdinal
- CompareTo
- Equals

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.CompareOrdinal Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two String objects by evaluating the numeric values of the corresponding Char objects in each string.

Overloads

Expand table

CompareOrdinal(String, String)	Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.
CompareOrdinal(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.

CompareOrdinal(String, String)

Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.

C#

```
public static int CompareOrdinal (string? strA, string? strB);
```

Parameters

`strA` String

The first string to compare.

`strB` String

The second string to compare.

Returns

Int32

An integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	`strA` is less than `strB`.
Zero	`strA` and `strB` are equal.
Greater than zero	`strA` is greater than `strB`.

Examples

The following example performs and ordinal comparison of two strings that only differ in case.

C#

```
// Sample for String.CompareOrdinal(String, String)
using System;

class Sample {
    public static void Main() {
    String str1 = "ABCD";
    String str2 = "abcd";
    String str;
    int result;

    Console.WriteLine();
    Console.WriteLine("Compare the numeric values of the corresponding
Char objects in each string.");
    Console.WriteLine("str1 = '{0}', str2 = '{1}'", str1, str2);
    result = String.CompareOrdinal(str1, str2);
    str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" :
"equal to"));
    Console.Write("String '{0}' is ", str1);
    Console.Write("{0} ", str);
    Console.WriteLine("String '{0}'.", str2);
    }
}
/*
This example produces the following results:

Compare the numeric values of the corresponding Char objects in each
string.
str1 = 'ABCD', str2 = 'abcd'
```

```
String 'ABCD' is less than String 'abcd'.
*/
```

Remarks

This method performs a case-sensitive comparison using ordinal sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions. To perform a case-insensitive comparison using ordinal sort rules, call the Compare(String, String, StringComparison) method with the `comparisonType` argument set to StringComparison.OrdinalIgnoreCase.

Because CompareOrdinal(String, String) is a static method, `strA` and `strB` can be `null`. If both values are `null`, the method returns 0 (zero), which indicates that `strA` and `strB` are equal. If only one of the values is `null`, the method considers the non-null value to be greater.

See also

- Int32
- Compare(String, String)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareOrdinal(String, Int32, String, Int32, Int32)

Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.

```
C#

public static int CompareOrdinal (string? strA, int indexA, string? strB,
int indexB, int length);
```

Parameters

strA String

The first string to use in the comparison.

indexA Int32

The starting index of the substring in `strA`.

strB String

The second string to use in the comparison.

indexB Int32

The starting index of the substring in `strB`.

length Int32

The maximum number of characters in the substrings to compare.

Returns

Int32

A 32-bit signed integer that indicates the lexical relationship between the two comparands.

Expand table

Value	Condition
Less than zero	The substring in `strA` is less than the substring in `strB`.
Zero	The substrings are equal, or `length` is zero.
Greater than zero	The substring in `strA` is greater than the substring in `strB`.

Exceptions

ArgumentOutOfRangeException

`strA` is not `null` and `indexA` is greater than `strA`.Length.

-or-

`strB` is not `null` and `indexB` is greater than `strB`.Length.

-or-

`indexA`, `indexB`, or `length` is negative.

Examples

This following example demonstrates that CompareOrdinal and Compare use different sort orders.

C#

```
using System;
using System.Globalization;

class Test
{
    public static void Main(String[] args)
    {
    String strLow = "abc";
    String strCap = "ABC";
    String result = "equal to ";
    int x = 0;
    int pos = 1;

// The Unicode codepoint for 'b' is greater than the codepoint for 'B'.
    x = String.CompareOrdinal(strLow, pos, strCap, pos, 1);
    if (x < 0) result = "less than";
    if (x > 0) result = "greater than";
    Console.WriteLine("CompareOrdinal(\"{0}\"[{2}], \"{1}\"[{2}]):",
strLow, strCap, pos);
    Console.WriteLine("   '{0}' is {1} '{2}'", strLow[pos], result,
strCap[pos]);

// In U.S. English culture, 'b' is linguistically less than 'B'.
    x = String.Compare(strLow, pos, strCap, pos, 1, false, new
CultureInfo("en-US"));
    if (x < 0) result = "less than";
    else if (x > 0) result = "greater than";
    Console.WriteLine("Compare(\"{0}\"[{2}], \"{1}\"[{2}]):", strLow,
strCap, pos);
    Console.WriteLine("   '{0}' is {1} '{2}'", strLow[pos], result,
strCap[pos]);
```

```
    }
}
```

Remarks

The `indexA`, `indexB`, and `length` parameters must be nonnegative.

The number of characters compared is the lesser of the length of `strA` less `indexA`, the length of `strB` less `indexB`, and `length`.

This method performs a case-sensitive comparison using ordinal sort rules. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions. To perform a case-insensitive comparison using ordinal sort rules, call the Compare(String, Int32, String, Int32, Int32, StringComparison) method with the `comparisonType` argument set to StringComparison.OrdinalIgnoreCase.

Because CompareOrdinal(String, String) is a static method, `strA` and `strB` can be `null`. If both values are `null`, the method returns 0 (zero), which indicates that `strA` and `strB` are equal. If only one of the values is `null`, the method considers the non-null value to be greater.

See also

- Compare(String, String)
- CompareTo(Object)
- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance with a specified object or String and returns an integer that indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified object or String.

Overloads

Expand table

CompareTo(String)	Compares this instance with a specified String object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
CompareTo(Object)	Compares this instance with a specified Object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.

Remarks

Both overloads of the CompareTo method perform culture-sensitive and case-sensitive comparison. You cannot use this method to perform culture-insensitive or ordinal comparisons. For code clarity, we recommend that you avoid the CompareTo method and call the Compare method instead.

CompareTo(String)

Compares this instance with a specified String object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.

```
C#
```

```
public int CompareTo (string? strB);
```

Parameters

`strB` String

The string to compare with this instance.

Returns

Int32

A 32-bit signed integer that indicates whether this instance precedes, follows, or appears in the same position in the sort order as the `strB` parameter.

Expand table

Value	Condition
Less than zero	This instance precedes `strB`.
Zero	This instance has the same position in the sort order as `strB`.
Greater than zero	This instance follows `strB`. -or- `strB` is `null`.

Implements

CompareTo(T)

Examples

The following example uses the CompareTo method to compare the current string instance with another string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
```

```
      string strFirst = "Goodbye";
      string strSecond = "Hello";
      string strThird = "a small string";
      string strFourth = "goodbye";

      // Compare a string to itself.
      Console.WriteLine(CompareStrings(strFirst, strFirst));

      Console.WriteLine(CompareStrings(strFirst, strSecond));
      Console.WriteLine(CompareStrings(strFirst, strThird));

      // Compare a string to another string that varies only by case.
      Console.WriteLine(CompareStrings(strFirst, strFourth));
      Console.WriteLine(CompareStrings(strFourth, strFirst));
   }

   private static string CompareStrings( string str1, string str2 )
   {
      // Compare the values, using the CompareTo method on the first
string.
      int cmpVal = str1.CompareTo(str2);

       if (cmpVal == 0) // The strings are the same.
         return "The strings occur in the same position in the sort
order.";
      else if (cmpVal < 0)
         return "The first string precedes the second in the sort
order.";
      else
         return "The first string follows the second in the sort order.";
    }
}
// The example displays the following output:
//       The strings occur in the same position in the sort order.
//       The first string precedes the second in the sort order.
//       The first string follows the second in the sort order.
//       The first string follows the second in the sort order.
//       The first string precedes the second in the sort order.
```

The following example demonstrates generic and non-generic versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;
```

```
class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
```

```
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.
```

```
    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method performs a word (case-sensitive and culture-sensitive) comparison using the current culture. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

> ⊗ **Caution**
>
> The **CompareTo** method was designed primarily for use in sorting or alphabetizing operations. It should not be used when the primary purpose of

the method call is to determine whether two strings are equivalent. To determine whether two strings are equivalent, call the **Equals** method.

For more information about the behavior of this method, see the Remarks section of the Compare(String, String) method.

This method implements the System.IComparable<T> interface and performs slightly better than the String.CompareTo(Object) method, because it does not have to determine whether the `strB` argument is a mutable value type that must be boxed, and it does not have to cast its parameter from an Object to a String.

Notes to Callers

Character sets include ignorable characters. The CompareTo(String) method does not consider such characters when it performs a culture-sensitive comparison. For example, if the following code is run on the .NET Framework 4 or later, a comparison of "animal" with "ani-mal" (using a soft hyphen, or U+00AD) indicates that the two strings are equivalent.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string s1 = "ani\u00ADmal";
      string s2 = "animal";

      Console.WriteLine("Comparison of '{0}' and '{1}': {2}",
                        s1, s2, s1.CompareTo(s2));
   }
}
// The example displays the following output:
//       Comparison of 'ani-mal' and 'animal': 0
```

To recognize ignorable characters in a string comparison, call the CompareOrdinal(String, String) method.

See also

- Compare(String, String)
- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares this instance with a specified Object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object that evaluates to a String.

Returns

Int32

A 32-bit signed integer that indicates whether this instance precedes, follows, or appears in the same position in the sort order as the `value` parameter.

Expand table

Value	Condition
Less than zero	This instance precedes `value`.

Value	Condition
Zero	This instance has the same position in the sort order as `value`.
Greater than zero	This instance follows `value`. -or- `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a String.

Examples

The following example uses the CompareTo method with an Object. Because it attempts to compare a String instance to a `TestClass` object, the method throws an ArgumentException.

C#

```
using System;

public class TestClass
{}

public class Example
{
   public static void Main()
   {
      var test = new TestClass();
      Object[] objectsToCompare = { test, test.ToString(), 123,
                                    123.ToString(), "some text",
                                    "Some Text" };
      string s = "some text";
      foreach (var objectToCompare in objectsToCompare) {
         try {
            int i = s.CompareTo(objectToCompare);
            Console.WriteLine("Comparing '{0}' with '{1}': {2}",
                              s, objectToCompare, i);
         }
         catch (ArgumentException) {
```

```
            Console.WriteLine("Bad argument: {0} (type {1})",
                              objectToCompare,
                              objectToCompare.GetType().Name);
        }
    }
  }
}
// The example displays the following output:
//    Bad argument: TestClass (type TestClass)
//    Comparing 'some text' with 'TestClass': -1
//    Bad argument: 123 (type Int32)
//    Comparing 'some text' with '123': 1
//    Comparing 'some text' with 'some text': 0
//    Comparing 'some text' with 'Some Text': -1
```

Remarks

`value` must be a String object.

> ⊗ **Caution**
>
> The **CompareTo** method was designed primarily for use in sorting or alphabetizing operations. It should not be used when the primary purpose of the method call is to determine whether two strings are equivalent. To determine whether two strings are equivalent, call the **Equals** method.

This method performs a word (case-sensitive and culture-sensitive) comparison using the current culture. For more information about word, string, and ordinal sorts, see System.Globalization.CompareOptions.

For more information about the behavior of this method, see the Remarks section of the String.Compare(String, String) method.

Notes to Callers

Character sets include ignorable characters. The CompareTo(Object) method does not consider such characters when it performs a culture-sensitive comparison. For example, if the following code is run on the .NET Framework 4 or later, a comparison of "animal" with "ani-mal" (using a soft hyphen, or U+00AD) indicates that the two strings are equivalent.

```
C#
```

```
using System;

public class Example
{
   public static void Main()
   {
      string s1 = "ani\u00ADmal";
      object o1 = "animal";

      Console.WriteLine("Comparison of '{0}' and '{1}': {2}",
                        s1, o1, s1.CompareTo(o1));
   }
}
// The example displays the following output:
//       Comparison of 'ani-mal' and 'animal': 0
```

To recognize ignorable characters in a string comparison, call the CompareOrdinal(String, String) method.

See also

- Compare(String, String)
- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

String.Concat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Concatenates one or more instances of String, or the String representations of the values of one or more instances of Object.

Overloads

Expand table

Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of four specified read-only character spans.
Concat(String, String, String, String)	Concatenates four specified instances of String.
Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of three specified read-only character spans.
Concat(Object, Object, Object)	Concatenates the string representations of three specified objects.
Concat(String, String)	Concatenates two specified instances of String.
Concat(String, String, String)	Concatenates three specified instances of String.
Concat(Object, Object)	Concatenates the string representations of two specified objects.
Concat(String[])	Concatenates the elements of a specified String array.
Concat(Object[])	Concatenates the string representations of the elements in a specified Object array.

Concat(Object)	Creates the string representation of a specified object.
Concat(IEnumerable<String>)	Concatenates the members of a constructed IEnumerable<T> collection of type String.
Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>)	Concatenates the string representations of two specified read-only character spans.
Concat<T>(IEnumerable<T>)	Concatenates the members of an IEnumerable<T> implementation.

Remarks

> **Note**
>
> You can also use your language's string concatenation operator, such as `+` in C# and F#, or `&` and `+` in Visual Basic, to concatenate strings. Both compilers translate the concatenation operator into a call to one of the overloads of `String.Concat`.

Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Concatenates the string representations of four specified read-only character spans.

C#

```
public static string Concat (ReadOnlySpan<char> str0, ReadOnlySpan<char>
str1, ReadOnlySpan<char> str2, ReadOnlySpan<char> str3);
```

Parameters

str0 ReadOnlySpan<Char>

The first read-only character span to concatenate.

str1 ReadOnlySpan<Char>

The second read-only character span to concatenate.

`str2` ReadOnlySpan<Char>

The third read-only character span to concatenate.

`str3` ReadOnlySpan<Char>

The fourth read-only character span to concatenate.

Returns

String

The concatenated string representations of the values of `str0`, `str1`, `str2` and `str3`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Concat(String, String, String, String)

Concatenates four specified instances of String.

C#

```
public static string Concat (string? str0, string? str1, string? str2,
string? str3);
```

Parameters

`str0` String

The first string to concatenate.

`str1` String

The second string to concatenate.

`str2` String

The third string to concatenate.

`str3` String

The fourth string to concatenate.

Returns

String

The concatenation of `str0`, `str1`, `str2`, and `str3`.

Examples

The following example defines an array of four-letter words and stores their individual letters to a string array in order to scramble them. It then calls the Concat(String, String, String, String) method to reassemble the scrambled words.

C#

```
using System;
using System.Collections;

public class Example
{
   public static void Main()
   {
      const int WORD_SIZE = 4;

      // Define some 4-letter words to be scrambled.
      string[] words = { "home", "food", "game", "rest" };
      // Define two arrays equal to the number of letters in each word.
      double[] keys = new double[WORD_SIZE];
      string[] letters = new string[WORD_SIZE];
      // Initialize the random number generator.
      Random rnd = new Random();

      // Scramble each word.
      foreach (string word in words)
      {
         for (int ctr = 0; ctr < word.Length; ctr++)
         {
            // Populate the array of keys with random numbers.
            keys[ctr] = rnd.NextDouble();
            // Assign a letter to the array of letters.
            letters[ctr] = word[ctr].ToString();
         }
         // Sort the array.
         Array.Sort(keys, letters, 0, WORD_SIZE, Comparer.Default);
         // Display the scrambled word.
         string scrambledWord = String.Concat(letters[0], letters[1],
                                              letters[2], letters[3]);
         Console.WriteLine("{0} --> {1}", word, scrambledWord);
      }
   }
```

```
}
// The example displays output like the following:
//       home --> mheo
//       food --> oodf
//       game --> aemg
//       rest --> trse
```

Remarks

The method concatenates `str0`, `str1`, `str2`, and `str3`; it does not add any delimiters.

See also

- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Concatenates the string representations of three specified read-only character spans.

C#

```
public static string Concat (ReadOnlySpan<char> str0, ReadOnlySpan<char>
str1, ReadOnlySpan<char> str2);
```

Parameters

`str0` ReadOnlySpan<Char>

The first read-only character span to concatenate.

`str1` ReadOnlySpan<Char>

The second read-only character span to concatenate.

`str2` ReadOnlySpan<Char>

The third read-only character span to concatenate.

Returns

String

The concatenated string representations of the values of `str0`, `str1` and `str2`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Concat(Object, Object, Object)

Concatenates the string representations of three specified objects.

C#

```
public static string Concat (object? arg0, object? arg1, object? arg2);
```

Parameters

`arg0` Object

The first object to concatenate.

`arg1` Object

The second object to concatenate.

`arg2` Object

The third object to concatenate.

Returns

String

The concatenated string representations of the values of `arg0`, `arg1`, and `arg2`.

Examples

The following example demonstrates the Concat method.

C#

```
using System;

class stringConcat5 {
    public static void Main() {
    int i = -123;
    Object o = i;
    Object[] objs = new Object[] {-123, -456, -789};

    Console.WriteLine("Concatenate 1, 2, and 3 objects:");
    Console.WriteLine("1) {0}", String.Concat(o));
    Console.WriteLine("2) {0}", String.Concat(o, o));
    Console.WriteLine("3) {0}", String.Concat(o, o, o));

    Console.WriteLine("\nConcatenate 4 objects and a variable length
parameter list:");
    Console.WriteLine("4) {0}", String.Concat(o, o, o, o));
    Console.WriteLine("5) {0}", String.Concat(o, o, o, o, o));

    Console.WriteLine("\nConcatenate a 3-element object array:");
    Console.WriteLine("6) {0}", String.Concat(objs));
    }
}
// The example displays the following output:
//    Concatenate 1, 2, and 3 objects:
//    1) -123
//    2) -123-123
//    3) -123-123-123
//
//    Concatenate 4 objects and a variable length parameter list:
//    4) -123-123-123-123
//    5) -123-123-123-123-123
//
//    Concatenate a 3-element object array:
//    6) -123-456-789
```

Remarks

The method concatenates `arg0`, `arg1`, and `arg2` by calling the parameterless `ToString` method of each object; it does not add any delimiters.

String.Empty is used in place of any null argument.

See also

- Object
- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(String, String)

Concatenates two specified instances of String.

C#

```
public static string Concat (string? str0, string? str1);
```

Parameters

`str0` String

The first string to concatenate.

`str1` String

The second string to concatenate.

Returns

String

The concatenation of `str0` and `str1`.

Examples

The following example concatenates a person's first, middle, and last name.

C#

```
using System;

public class ConcatTest {
    public static void Main() {

        // we want to simply quickly add this person's name together
        string fName = "Simon";
        string mName = "Jake";
        string lName = "Harrows";

        // because we want a name to appear with a space in between each
name,
        // put a space on the front of the middle, and last name,
allowing for
        // the fact that a space may already be there
        mName = " " + mName.Trim();
        lName = " " + lName.Trim();

        // this line simply concatenates the two strings
        Console.WriteLine("Welcome to this page, '{0}'!", string.Concat(
string.Concat(fName, mName), lName ) );
    }
}
// The example displays the following output:
//        Welcome to this page, 'Simon Jake Harrows'!
```

Remarks

The method concatenates `str0` and `str1`; it does not add any delimiters.

An Empty string is used in place of any null argument.

See also

- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(String, String, String)

Concatenates three specified instances of String.

C#

```
public static string Concat (string? str0, string? str1, string? str2);
```

Parameters

`str0` String

The first string to concatenate.

`str1` String

The second string to concatenate.

`str2` String

The third string to concatenate.

Returns

String

The concatenation of `str0`, `str1`, and `str2`.

Examples

The following example uses the Concat method to concatenate three strings and displays the result.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String s1 = "We went to a bookstore, ";
      String s2 = "a movie, ";
      String s3 = "and a restaurant.";

      var s = String.Concat(s1, s2, s3);
      Console.WriteLine(s);
   }
}
// The example displays the following output:
//      We went to a bookstore, a movie, and a restaurant.
```

Remarks

The method concatenates `str0`, `str1`, and `str2`; it does not add any delimiters.

See also

- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Concat(Object, Object)

Concatenates the string representations of two specified objects.

C#

```
public static string Concat (object? arg0, object? arg1);
```

Parameters

`arg0` Object

The first object to concatenate.

`arg1` Object

The second object to concatenate.

Returns

String

The concatenated string representations of the values of `arg0` and `arg1`.

Examples

The following example demonstrates the Concat method.

C#

```
using System;

class stringConcat5 {
    public static void Main() {
    int i = -123;
    Object o = i;
    Object[] objs = new Object[] {-123, -456, -789};

    Console.WriteLine("Concatenate 1, 2, and 3 objects:");
    Console.WriteLine("1) {0}", String.Concat(o));
    Console.WriteLine("2) {0}", String.Concat(o, o));
    Console.WriteLine("3) {0}", String.Concat(o, o, o));
```

```
    Console.WriteLine("\nConcatenate 4 objects and a variable length
parameter list:");
    Console.WriteLine("4) {0}", String.Concat(o, o, o, o));
    Console.WriteLine("5) {0}", String.Concat(o, o, o, o, o));

    Console.WriteLine("\nConcatenate a 3-element object array:");
    Console.WriteLine("6) {0}", String.Concat(objs));
    }
}
// The example displays the following output:
//    Concatenate 1, 2, and 3 objects:
//    1) -123
//    2) -123-123
//    3) -123-123-123
//
//    Concatenate 4 objects and a variable length parameter list:
//    4) -123-123-123-123
//    5) -123-123-123-123-123
//
//    Concatenate a 3-element object array:
//    6) -123-456-789
```

Remarks

The method concatenates `arg0` and `arg1` by calling the parameterless `ToString` method of `arg0` and `arg1`; it does not add any delimiters.

String.Empty is used in place of any null argument.

If either of the arguments is an array reference, the method concatenates a string representing that array, instead of its members (for example, "System.String[]").

See also

- Object
- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(String[])

> Important
>
> This API is not CLS-compliant.

Concatenates the elements of a specified String array.

C#

```
public static string Concat (params string?[] values);
```

Parameters

`values` String[]

An array of string instances.

Returns

String

The concatenated elements of `values`.

Exceptions

ArgumentNullException

`values` is `null`.

OutOfMemoryException

Out of memory.

Examples

The following example demonstrates the use of the Concat method with a String array.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        // Make an array of strings. Note that we have included spaces.
        string [] s = { "hello ", "and ", "welcome ", "to ",
                        "this ", "demo! " };

        // Put all the strings together.
        Console.WriteLine(string.Concat(s));

        // Sort the strings, and put them together.
        Array.Sort(s);
        Console.WriteLine(string.Concat(s));
    }
}
// The example displays the following output:
//       hello and welcome to this demo!
//       and demo! hello this to welcome
```

Remarks

The method concatenates each object in `values`; it does not add any delimiters.

An Empty string is used in place of any null object in the array.

See also

- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(Object[])

Concatenates the string representations of the elements in a specified Object array.

```
C#
public static string Concat (params object?[] args);
```

Parameters

`args` Object[]

An object array that contains the elements to concatenate.

Returns

String

The concatenated string representations of the values of the elements in `args`.

Exceptions

ArgumentNullException

`args` is `null`.

OutOfMemoryException

Out of memory.

Examples

The following example demonstrates the use of the Concat method with an Object array.

```
C#
```

```
using System;

public class ConcatTest {
    public static void Main() {
        // Create a group of objects.
        Test1 t1 = new Test1();
        Test2 t2 = new Test2();
        int i = 16;
        string s = "Demonstration";

        // Place the objects in an array.
        object [] o = { t1, i, t2, s };

        // Concatenate the objects together as a string. To do this,
        // the ToString method of each of the objects is called.
        Console.WriteLine(string.Concat(o));
    }
}

// Create two empty test classes.
class Test1 {
}

class Test2 {
}
// The example displays the following output:
//       Test116Test2Demonstration
```

Remarks

The method concatenates each object in `args` by calling the parameterless `ToString` method of that object; it does not add any delimiters.

String.Empty is used in place of any null object in the array.

Notes to Callers

This method is not called by C++ code. The C++ compiler resolves calls to Concat that have four or more object parameters as a call to Concat(Object, Object, Object, Object).

See also

- Object
- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(Object)

Creates the string representation of a specified object.

C#

```
public static string Concat (object? arg0);
```

Parameters

`arg0` Object

The object to represent, or `null`.

Returns

String

The string representation of the value of `arg0`, or Empty if `arg0` is `null`.

Examples

The following example demonstrates the Concat method.

C#

```
using System;

class stringConcat5 {
```

```
    public static void Main() {
    int i = -123;
    Object o = i;
    Object[] objs = new Object[] {-123, -456, -789};

    Console.WriteLine("Concatenate 1, 2, and 3 objects:");
    Console.WriteLine("1) {0}", String.Concat(o));
    Console.WriteLine("2) {0}", String.Concat(o, o));
    Console.WriteLine("3) {0}", String.Concat(o, o, o));

    Console.WriteLine("\nConcatenate 4 objects and a variable length
parameter list:");
    Console.WriteLine("4) {0}", String.Concat(o, o, o, o));
    Console.WriteLine("5) {0}", String.Concat(o, o, o, o, o));

    Console.WriteLine("\nConcatenate a 3-element object array:");
    Console.WriteLine("6) {0}", String.Concat(objs));
    }
}
// The example displays the following output:
//    Concatenate 1, 2, and 3 objects:
//    1) -123
//    2) -123-123
//    3) -123-123-123
//
//    Concatenate 4 objects and a variable length parameter list:
//    4) -123-123-123-123
//    5) -123-123-123-123-123
//
//    Concatenate a 3-element object array:
//    6) -123-456-789
```

Remarks

The Concat(Object) method represents `arg0` as a string by calling its parameterless `ToString` method.

See also

- Object
- Join(String, String[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(IEnumerable<String>)

Concatenates the members of a constructed IEnumerable<T> collection of type String.

C#

```
public static string Concat
(System.Collections.Generic.IEnumerable<string?> values);
```

Parameters

values IEnumerable<String>

A collection object that implements IEnumerable<T> and whose generic type argument is String.

Returns

String

The concatenated strings in `values`, or Empty if `values` is an empty `IEnumerable(Of String)`.

Exceptions

ArgumentNullException

`values` is `null`.

Examples

The following example uses the Sieve of Eratosthenes algorithm to calculate the prime numbers that are less than or equal to 100. It assigns the result to a List<T> object of type String, which it then passes to the Concat(IEnumerable<String>) method.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      int maxPrime = 100;
      IEnumerable<String> primeList = GetPrimes(maxPrime);
      Console.WriteLine("Primes less than {0}:", maxPrime);
      Console.WriteLine("   {0}", String.Concat(primeList));
   }

   private static IEnumerable<String> GetPrimes(int maxPrime)
   {
      Array values = Array.CreateInstance(typeof(int),
                                 new int[] { maxPrime - 1}, new int[] { 2
});
      // Use Sieve of Erathsthenes to determine prime numbers.
      for (int ctr = values.GetLowerBound(0); ctr <= (int)
Math.Ceiling(Math.Sqrt(values.GetUpperBound(0))); ctr++)
      {

         if ((int) values.GetValue(ctr) == 1) continue;

         for (int multiplier = ctr; multiplier <=  maxPrime / 2;
multiplier++)
            if (ctr * multiplier <= maxPrime)
               values.SetValue(1, ctr * multiplier);
      }

      List<String> primes = new List<String>();
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++)
         if ((int) values.GetValue(ctr) == 0)
            primes.Add(ctr.ToString() + " ");
      return primes;
   }
}
// The example displays the following output:
//    Primes less than 100:
//       2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
```

Remarks

The method concatenates each object in `values`; it does not add any delimiters. To specify a delimiter between each member of `values`, call the Join(String, IEnumerable<String>) method.

An Empty string is used in place of any null element in `values`.

If `values` is an empty `IEnumerable(Of String)`, the method returns String.Empty. If `values` is `null`, the method throws an ArgumentNullException exception.

Concat(IEnumerable<String>) is a convenience method that lets you concatenate each element in an `IEnumerable(Of String)` collection without first converting the elements to a string array. It is particularly useful with Language-Integrated Query (LINQ) query expressions. The following example passes a `List(Of String)` object that contains either the uppercase or lowercase letters of the alphabet to a lambda expression that selects letters that are equal to or greater than a particular letter (which, in the example, is "M"). The `IEnumerable(Of String)` collection that is returned by the Enumerable.Where method is passed to the Concat(IEnumerable<String>) method to display the result as a single string.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Example
{
   public static void Main()
   {
      string output = String.Concat( GetAlphabet(true).Where( letter =>
                      letter.CompareTo("M") >= 0));
      Console.WriteLine(output);
   }

   private static List<string> GetAlphabet(bool upper)
   {
      List<string> alphabet = new List<string>();
      int charValue = upper ? 65 : 97;
      for (int ctr = 0; ctr <= 25; ctr++)
         alphabet.Add(((char)(charValue + ctr)).ToString());
      return alphabet;
   }
}
// The example displays the following output:
//      MNOPQRSTUVWXYZ
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Concat(ReadOnlySpan<Char>, ReadOnlySpan<Char>)

Concatenates the string representations of two specified read-only character spans.

C#

```
public static string Concat (ReadOnlySpan<char> str0, ReadOnlySpan<char>
str1);
```

Parameters

`str0` ReadOnlySpan<Char>

The first read-only character span to concatenate.

`str1` ReadOnlySpan<Char>

The second read-only character span to concatenate.

Returns

String

The concatenated string representations of the values of `str0` and `str1`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Concat<T>(IEnumerable<T>)

Concatenates the members of an IEnumerable<T> implementation.

```
C#

public static string Concat<T> (System.Collections.Generic.IEnumerable<T>
values);
```

Type Parameters

T

The type of the members of `values`.

Parameters

values IEnumerable<T>

A collection object that implements the IEnumerable<T> interface.

Returns

String

The concatenated members in `values`.

Exceptions

ArgumentNullException

`values` is `null`.

Examples

The following example defines a very simple `Animal` class that contains the name of an animal and the order to which it belongs. It then defines a List<T> object to contain a number of `Animal` objects. The Enumerable.Where extension method is called to extract the `Animal` objects whose `Order` property equals "Rodent". The

result is passed to the Concat<T>(IEnumerable<T>) method and displayed to the console.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Animal
{
   public string Kind;
   public string Order;

   public Animal(string kind, string order)
   {
      this.Kind = kind;
      this.Order = order;
   }

   public override string ToString()
   {
      return this.Kind;
   }
}

public class Example
{
   public static void Main()
   {
      List<Animal> animals = new List<Animal>();
      animals.Add(new Animal("Squirrel", "Rodent"));
      animals.Add(new Animal("Gray Wolf", "Carnivora"));
      animals.Add(new Animal("Capybara", "Rodent"));
      string output = String.Concat(animals.Where( animal =>
                      (animal.Order == "Rodent")));
      Console.WriteLine(output);
   }
}
// The example displays the following output:
//      SquirrelCapybara
```

Remarks

The method concatenates each object in `values`; it does not add any delimiters.

An Empty string is used in place of any null argument.

Concat<T>(IEnumerable<T>) is a convenience method that lets you concatenate each element in an IEnumerable<T> collection without first converting the elements

to strings. It is particularly useful with Language-Integrated Query (LINQ) query expressions, as the example illustrates. The string representation of each object in the IEnumerable<T> collection is derived by calling that object's `ToString` method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Contains Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Contains(Char)	Returns a value indicating whether a specified character occurs within this string.
Contains(String)	Returns a value indicating whether a specified substring occurs within this string.
Contains(Char, StringComparison)	Returns a value indicating whether a specified character occurs within this string, using the specified comparison rules.
Contains(String, StringComparison)	Returns a value indicating whether a specified string occurs within this string, using the specified comparison rules.

Contains(Char)

Returns a value indicating whether a specified character occurs within this string.

C#

```
public bool Contains (char value);
```

Parameters

`value` Char

The character to seek.

Returns

Boolean

`true` if the `value` parameter occurs within this string; otherwise, `false`.

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Contains(String)

Returns a value indicating whether a specified substring occurs within this string.

C#

```
public bool Contains (string value);
```

Parameters

`value` String

The string to seek.

Returns

Boolean

`true` if the `value` parameter occurs within this string, or if `value` is the empty string (""); otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example determines whether the string "fox" is a substring of a familiar quotation. If "fox" is found in the string, it also displays its starting position.

C#

```
string s1 = "The quick brown fox jumps over the lazy dog";
string s2 = "fox";
bool b = s1.Contains(s2);
Console.WriteLine("'{0}' is in the string '{1}': {2}",
                s2, s1, b);
if (b) {
    int index = s1.IndexOf(s2);
    if (index >= 0)
        Console.WriteLine("'{0} begins at character position {1}",
                      s2, index + 1);
}
// This example displays the following output:
//    'fox' is in the string 'The quick brown fox jumps over the lazy
dog': True
//    'fox begins at character position 17
```

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison. The search begins at the first character position of this string and continues through the last character position.

To perform a culture-sensitive or ordinal case-insensitive comparison:

- On .NET Core 2.1 and later versions: Call the Contains(String, StringComparison) overload instead.
- On .NET Framework: Create a custom method. The following example illustrates one such approach. It defines a String extension method that includes a StringComparison parameter and indicates whether a string contains a substring when using the specified form of string comparison.

C#

```
using System;

public static class StringExtensions
{
   public static bool Contains(this String str, String substring,
                               StringComparison comp)
   {
```

```
        if (substring == null)
            throw new ArgumentNullException("substring",
                                            "substring cannot be null.");
        else if (! Enum.IsDefined(typeof(StringComparison), comp))
            throw new ArgumentException("comp is not a member of
StringComparison",
                                        "comp");

        return str.IndexOf(substring, comp) >= 0;
    }
}
```

C#

```
String s = "This is a string.";
String sub1 = "this";
Console.WriteLine("Does '{0}' contain '{1}'?", s, sub1);
StringComparison comp = StringComparison.Ordinal;
Console.WriteLine("   {0:G}: {1}", comp, s.Contains(sub1, comp));

comp = StringComparison.OrdinalIgnoreCase;
Console.WriteLine("   {0:G}: {1}", comp, s.Contains(sub1, comp));

// The example displays the following output:
//       Does 'This is a string.' contain 'this'?
//          Ordinal: False
//          OrdinalIgnoreCase: True
```

If you are interested in the position of the substring `value` in the current instance, you can call the IndexOf method to get the starting position of its first occurrence, or you can call the LastIndexOf method to get the starting position of its last occurrence. The example includes a call to the IndexOf(String) method if a substring is found in a string instance.

See also

- IndexOf(Char)
- LastIndexOf(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Contains(Char, StringComparison)

Returns a value indicating whether a specified character occurs within this string, using the specified comparison rules.

C#

```
public bool Contains (char value, StringComparison comparisonType);
```

Parameters

`value` Char

The character to seek.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Boolean

`true` if the `value` parameter occurs within this string; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Contains(String, StringComparison)

Returns a value indicating whether a specified string occurs within this string, using the specified comparison rules.

```
C#
public bool Contains (string value, StringComparison comparisonType);
```

Parameters

`value` String

The string to seek.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Boolean

`true` if the `value` parameter occurs within this string, or if `value` is the empty string (""); otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

String.Copy(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API should not be used to create mutable strings. See **https://go.microsoft.com/fwlink/?linkid=2084035** for alternatives.

Creates a new instance of String with the same value as a specified String.

C#

```
[System.Obsolete("This API should not be used to create mutable strings. See https://go.microsoft.com/fwlink/?linkid=2084035 for alternatives.")]
public static string Copy (string str);
```

Parameters

`str` String

The string to copy.

Returns

String

A new string with the same value as `str`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`str` is `null`.

Remarks

The `Copy` method returns a String object that has the same value as the original string but represents a different object reference. It differs from an assignment operation, which assigns an existing string reference to an additional object variable.

> **ⓘ Important**
>
> Starting with .NET Core 3.0, this method is obsolete. However, we do not recommend its use in any .NET implementation. In particular, because of changes in string interning in .NET Core 3.0, in some cases the `Copy` method will not create a new string but will simply return a reference to an existing interned string.

Depending on Why you want to call the `Copy` method, there are a number of alternatives:

- If you want a different string instance to use in an operation that modifies the string, use the original string instance. Because strings are immutable, the string operation creates a new string instance, and the original string remains unaffected. In this case, you should not assign the new string reference to the original string variable. The following example provides an illustration.

  C#

  ```
  var original = "This is a sentence. This is a second sentence.";
  var sentence1 = original.Substring(0, original.IndexOf(".") + 1);
  Console.WriteLine(original);
  Console.WriteLine(sentence1);
  // The example displays the following output:
  //    This is a sentence. This is a second sentence.
  //    This is a sentence.
  ```

  In this case, calling the `Copy` method to create a new string before calling the Substring method unnecessarily creates a new string instance.

- If you want to create a mutable buffer with the same contents as the original string, call the String.ToCharArray or StringBuilder.StringBuilder(String) constructor. For example:

  C#

  ```
  private static void UseMutableBuffer()
  {
      var original = "This is a sentence. This is a second sentence.";
  ```

```
    var chars = original.ToCharArray();
    var span = new Span<char>(chars);
    var slice = span.Slice(span.IndexOf('.'), 3);
    slice = MergeSentence(slice);
    Console.WriteLine($"Original string: {original}");
    Console.WriteLine($"Modified string: {span.ToString()}");

    static Span<char> MergeSentence(Span<char> span)
    {
        if (span.Length == 0) return Span<char>.Empty;

        span[0] = ';';
        span[2] = Char.ToLower(span[2]);
        return span;
    }
}
// The example displays the following output:
//    Original string: This is a sentence. This is a second sentence.
//    Modified string: This is a sentence; this is a second sentence.
```

- If you want to create a mutable copy of the string so that you can use unsafe code to modify the string contents, use Marshal.StringToHGlobalUni method. The following example uses the Marshal.StringToHGlobalUni method to get a pointer to the location of an copied string in unmanaged memory, increments the Unicode code point of each character in the string by one, and copies the resulting string back to a managed string.

C#

```
private static void UseUnmanaged()
{
    var original = "This is a single sentence.";
    var len = original.Length;
    var ptr = Marshal.StringToHGlobalUni(original);
    string? result;
    unsafe
    {
        char *ch = (char *) ptr.ToPointer();
        while (len-- > 0)
        {
            char c = Convert.ToChar(Convert.ToUInt16(*ch) + 1);
            *ch++ = c;
        }
        result = Marshal.PtrToStringUni(ptr);
        Marshal.FreeHGlobal(ptr);
    }
    Console.WriteLine($"Original string: {original}");
    Console.WriteLine($"String from interop: '{result}'");
}
// The example displays the following output:
```

```
//    Original string: This is a single sentence.
//    String from interop: 'Uijt!jt!b!tjohmf!tfoufodf/'
```

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2 *(Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.CopyTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CopyTo(Span<Char>)	Copies the contents of this string into the destination span.
CopyTo(Int32, Char[], Int32, Int32)	Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.

CopyTo(Span<Char>)

Copies the contents of this string into the destination span.

C#

```
public void CopyTo (Span<char> destination);
```

Parameters

`destination` Span<Char>

The span into which to copy this string's contents.

Exceptions

ArgumentException

The destination span is shorter than the source string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

CopyTo(Int32, Char[], Int32, Int32)

Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.

```
C#

public void CopyTo (int sourceIndex, char[] destination, int
destinationIndex, int count);
```

Parameters

`sourceIndex` Int32

The index of the first character in this instance to copy.

`destination` Char[]

An array of Unicode characters to which characters in this instance are copied.

`destinationIndex` Int32

The index in `destination` at which the copy operation begins.

`count` Int32

The number of characters in this instance to copy to `destination`.

Exceptions

ArgumentNullException

`destination` is `null`.

ArgumentOutOfRangeException

`sourceIndex`, `destinationIndex`, or `count` is negative

-or-

`sourceIndex` does not identify a position in the current instance.

-or-

`destinationIndex` does not identify a valid index in the `destination` array.

-or-

`count` is greater than the length of the substring from `sourceIndex` to the end of this instance

-or-

`count` is greater than the length of the subarray from `destinationIndex` to the end of the `destination` array.

Examples

The following example demonstrates the CopyTo method.

C#

```
using System;

public class CopyToTest {
    public static void Main() {

        // Embed an array of characters in a string
        string strSource = "changed";
    char [] destination = { 'T', 'h', 'e', ' ', 'i', 'n', 'i', 't', 'i',
'a', 'l', ' ',
                'a', 'r', 'r', 'a', 'y' };

        // Print the char array
        Console.WriteLine( destination );

        // Embed the source string in the destination string
        strSource.CopyTo ( 0, destination, 4, strSource.Length );

        // Print the resulting array
        Console.WriteLine( destination );

        strSource = "A different string";

        // Embed only a section of the source string in the destination
        strSource.CopyTo ( 2, destination, 3, 9 );

        // Print the resulting array
        Console.WriteLine( destination );
    }
}
// The example displays the following output:
//       The initial array
```

```
//        The changed array
//        Thedifferentarray
```

Remarks

This method copies `count` characters from the `sourceIndex` position of this instance to the `destinationIndex` position of `destination` character array. This method does not resize the `destination` character array; it must have a sufficient number of elements to accommodate the copied characters or the method throws an ArgumentOutOfRangeException.

`sourceIndex` and `destinationIndex` are zero-based.

See also

- Char
- Int32
- Insert(Int32, String)
- Substring(Int32)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

String.Create Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Create(IFormatProvider, Span<Char>, DefaultInterpolatedStringHandler)	Creates a new string by using the specified provider to control the formatting of the specified interpolated string.
Create(IFormatProvider, DefaultInterpolatedStringHandler)	Creates a new string by using the specified provider to control the formatting of the specified interpolated string.
Create<TState>(Int32, TState, SpanAction<Char,TState>)	Creates a new string with a specific length and initializes it after creation by using the specified callback.

Create(IFormatProvider, Span<Char>, DefaultInterpolatedStringHandler)

Creates a new string by using the specified provider to control the formatting of the specified interpolated string.

C#

```
public static string Create (IFormatProvider? provider, Span<char>
initialBuffer, ref
System.Runtime.CompilerServices.DefaultInterpolatedStringHandler
handler);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

`initialBuffer` Span<Char>

The initial buffer that may be used as temporary space as part of the formatting operation. The contents of this buffer may be overwritten.

`handler` DefaultInterpolatedStringHandler

The interpolated string, passed by reference.

Returns

String

The string that results for formatting the interpolated string using the specified format provider.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Create(IFormatProvider, DefaultInterpolatedStringHandler)

Creates a new string by using the specified provider to control the formatting of the specified interpolated string.

C#

```
public static string Create (IFormatProvider? provider, ref
System.Runtime.CompilerServices.DefaultInterpolatedStringHandler
handler);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`handler` DefaultInterpolatedStringHandler

The interpolated string, passed by reference.

Returns

String

The string that results for formatting the interpolated string using the specified format provider.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Create<TState>(Int32, TState, SpanAction<Char,TState>)

Creates a new string with a specific length and initializes it after creation by using the specified callback.

C#

```
public static string Create<TState> (int length, TState state,
System.Buffers.SpanAction<char,TState> action);
```

Type Parameters

`TState`

The type of the element to pass to `action`.

Parameters

`length` Int32

The length of the string to create.

`state` TState

The element to pass to `action`.

`action` SpanAction<Char,TState>

A callback to initialize the string.

Returns

String

The created string.

Remarks

The initial content of the destination span passed to `action` is undefined. Therefore, it is the delegate's responsibility to ensure that every element of the span is assigned. Otherwise, the resulting string could contain random characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.EndsWith Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the end of this string instance matches a specified string.

Overloads

Expand table

EndsWith(Char)	Determines whether the end of this string instance matches the specified character.
EndsWith(String)	Determines whether the end of this string instance matches the specified string.
EndsWith(String, StringComparison)	Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.
EndsWith(String, Boolean, CultureInfo)	Determines whether the end of this string instance matches the specified string when compared using the specified culture.

EndsWith(Char)

Determines whether the end of this string instance matches the specified character.

C#

```
public bool EndsWith (char value);
```

Parameters

`value` Char

The character to compare to the character at the end of this instance.

Returns

Boolean

`true` if `value` matches the end of this instance; otherwise, `false`.

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

EndsWith(String)

Determines whether the end of this string instance matches the specified string.

C#

```
public bool EndsWith (string value);
```

Parameters

`value` String

The string to compare to the substring at the end of this instance.

Returns

Boolean

`true` if `value` matches the end of this instance; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example indicates whether each string in an array ends with a period (".").

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String[] strings = { "This is a string.", "Hello!", "Nothing.",
                           "Yes.", "randomize" };
      foreach (var value in strings) {
         bool endsInPeriod = value.EndsWith(".");
         Console.WriteLine("'{0}' ends in a period: {1}",
                           value, endsInPeriod);
      }
   }
}
// The example displays the following output:
//       'This is a string.' ends in a period: True
//       'Hello!' ends in a period: False
//       'Nothing.' ends in a period: True
//       'Yes.' ends in a period: True
//       'randomize' ends in a period: False
```

The following example defines a `StripEndTags` method that uses the EndsWith(String) method to remove HTML end tags from the end of a line. Note that the `StripEndTags` method is called recursively to ensure that multiple HTML end tags at the end of the line are removed.

C#

```
using System;

public class EndsWithTest {
    public static void Main() {

        // process an input file that contains html tags.
        // this sample checks for multiple tags at the end of the line,
rather than simply
        // removing the last one.
        // note: HTML markup tags always end in a greater than symbol
(>).

        string [] strSource = { "<b>This is bold text</b>", "<H1>This is
large Text</H1>",
```

```
                "<b><i><font color=green>This has multiple tags</font>
</i></b>",
                "<b>This has <i>embedded</i> tags.</b>",
                "This line simply ends with a greater than symbol, it
should not be modified>" };

        Console.WriteLine("The following lists the items before the ends
have been stripped:");
        Console.WriteLine("-----------------------------------------------
-------------------");

        // print out the initial array of strings
        foreach ( string s in strSource )
            Console.WriteLine( s );

        Console.WriteLine();

        Console.WriteLine("The following lists the items after the ends
have been stripped:");
        Console.WriteLine("-----------------------------------------------
------------------");

        // print out the array of strings
        foreach (var s in strSource)
            Console.WriteLine(StripEndTags(s));
    }

    private static string StripEndTags( string item ) {

        bool found = false;

        // try to find a tag at the end of the line using EndsWith
        if (item.Trim().EndsWith(">")) {

            // now search for the opening tag...
            int lastLocation = item.LastIndexOf( "</" );

            // remove the identified section, if it is a valid region
            if ( lastLocation >= 0 ) {
                found = true;
                item =  item.Substring( 0, lastLocation );
            }
        }

        if (found)
           item = StripEndTags(item);

        return item;
    }
}
// The example displays the following output:
//    The following lists the items before the ends have been stripped:
//    ----------------------------------------------------------------
//    <b>This is bold text</b>
//    <H1>This is large Text</H1>
```

```
//    <b><i><font color=green>This has multiple tags</font></i></b>
//    <b>This has <i>embedded</i> tags.</b>
//    This line simply ends with a greater than symbol, it should not be
modified>
//
//    The following lists the items after the ends have been stripped:
//    ---------------------------------------------------------------
//    <b>This is bold text
//    <H1>This is large Text
//    <b><i><font color=green>This has multiple tags
//    <b>This has <i>embedded</i> tags.
//    This line simply ends with a greater than symbol, it should not be
modified>
```

Remarks

This method compares `value` to the substring at the end of this instance that is the same length as `value`, and returns an indication whether they are equal. To be equal, `value` must be a reference to this same instance or match the end of this instance.

This method performs a word (case-sensitive and culture-sensitive) comparison using the current culture.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To determine whether a string ends with a particular substring by using the string comparison rules of the current culture, signal your intention explicitly by calling the EndsWith(String, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- StartsWith(String)
- Compare(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

EndsWith(String, StringComparison)

Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.

C#

```
public bool EndsWith (string value, StringComparison comparisonType);
```

Parameters

`value` String

The string to compare to the substring at the end of this instance.

`comparisonType` StringComparison

One of the enumeration values that determines how this string and `value` are compared.

Returns

Boolean

`true` if the `value` parameter matches the end of this string; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`comparisonType` is not a StringComparison value.

Examples

The following example determines whether a string ends with a particular substring. The results are affected by the choice of culture, whether case is ignored, and whether an ordinal comparison is performed.

C#

```
// This example demonstrates the
// System.String.EndsWith(String, StringComparison) method.

using System;
using System.Threading;

class Sample
{
    public static void Main()
    {
        string intro = "Determine whether a string ends with another
string, " +
                   "using\n  different values of StringComparison.";

        StringComparison[] scValues = {
            StringComparison.CurrentCulture,
            StringComparison.CurrentCultureIgnoreCase,
            StringComparison.InvariantCulture,
            StringComparison.InvariantCultureIgnoreCase,
            StringComparison.Ordinal,
            StringComparison.OrdinalIgnoreCase };

        Console.WriteLine(intro);

        // Display the current culture because the culture-specific
comparisons
        // can produce different results with different cultures.
        Console.WriteLine("The current culture is {0}.\n",
                       Thread.CurrentThread.CurrentCulture.Name);

        // Determine whether three versions of the letter I are equal to
each other.
        foreach (StringComparison sc in scValues)
        {
            Console.WriteLine("StringComparison.{0}:", sc);
            Test("abcXYZ", "XYZ", sc);
            Test("abcXYZ", "xyz", sc);
            Console.WriteLine();
        }
    }

    protected static void Test(string x, string y, StringComparison
```

```
comparison)
    {
        string resultFmt = "\"{0}\" {1} with \"{2}\".";
        string result = "does not end";

        if (x.EndsWith(y, comparison))
            result = "ends";
        Console.WriteLine(resultFmt, x, result, y);
    }
}

/*
This code example produces the following results:

Determine whether a string ends with another string, using
  different values of StringComparison.
The current culture is en-US.

StringComparison.CurrentCulture:
"abcXYZ" ends with "XYZ".
"abcXYZ" does not end with "xyz".

StringComparison.CurrentCultureIgnoreCase:
"abcXYZ" ends with "XYZ".
"abcXYZ" ends with "xyz".

StringComparison.InvariantCulture:
"abcXYZ" ends with "XYZ".
"abcXYZ" does not end with "xyz".

StringComparison.InvariantCultureIgnoreCase:
"abcXYZ" ends with "XYZ".
"abcXYZ" ends with "xyz".

StringComparison.Ordinal:
"abcXYZ" ends with "XYZ".
"abcXYZ" does not end with "xyz".

StringComparison.OrdinalIgnoreCase:
"abcXYZ" ends with "XYZ".
"abcXYZ" ends with "xyz".

*/
```

Remarks

The EndsWith method compares the `value` parameter to the substring at the end of this string and returns a value that indicates whether they are equal. To be equal, `value` must be a reference to this same string, must be the empty string ("""), or must match the end of this string. The type of comparison performed by the EndsWith method depends on the value of the `comparisonType` parameter.

See also

- StringComparison

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

EndsWith(String, Boolean, CultureInfo)

Determines whether the end of this string instance matches the specified string when compared using the specified culture.

C#

```
public bool EndsWith (string value, bool ignoreCase,
System.Globalization.CultureInfo? culture);
```

Parameters

`value` String

The string to compare to the substring at the end of this instance.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

`culture` CultureInfo

Cultural information that determines how this instance and `value` are compared. If `culture` is `null`, the current culture is used.

Returns

Boolean

`true` if the `value` parameter matches the end of this string; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example determines whether a string occurs at the end of another string. The EndsWith method is called several times using case sensitivity, case insensitivity, and different cultures that influence the results of the search.

C#

```
// This code example demonstrates the
// System.String.EndsWith(String, ..., CultureInfo) method.

using System;
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
        string msg1 = "Search for the target string \"{0}\" in the string
\"{1}\".\n";
        string msg2 = "Using the {0} - \"{1}\" culture:";
        string msg3 = "  The string to search ends with the target
string: {0}";
        bool result = false;
        CultureInfo ci;

        // Define a target string to search for.
        // U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
        string capitalARing = "\u00c5";

        // Define a string to search.
        // The result of combining the characters LATIN SMALL LETTER A
and COMBINING
        // RING ABOVE (U+0061, U+030a) is linguistically equivalent to
the character
        // LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
        string xyzARing = "xyz" + "\u0061\u030a";
```

```
        // Display the string to search for and the string to search.
        Console.WriteLine(msg1, capitalARing, xyzARing);

        // Search using English-United States culture.
        ci = new CultureInfo("en-US");
        Console.WriteLine(msg2, ci.DisplayName, ci.Name);

        Console.WriteLine("Case sensitive:");
        result = xyzARing.EndsWith(capitalARing, false, ci);
        Console.WriteLine(msg3, result);

        Console.WriteLine("Case insensitive:");
        result = xyzARing.EndsWith(capitalARing, true, ci);
        Console.WriteLine(msg3, result);
        Console.WriteLine();

        // Search using Swedish-Sweden culture.
        ci = new CultureInfo("sv-SE");
        Console.WriteLine(msg2, ci.DisplayName, ci.Name);

        Console.WriteLine("Case sensitive:");
        result = xyzARing.EndsWith(capitalARing, false, ci);
        Console.WriteLine(msg3, result);

        Console.WriteLine("Case insensitive:");
        result = xyzARing.EndsWith(capitalARing, true, ci);
        Console.WriteLine(msg3, result);
    }
}

/*
This code example produces the following results (for en-us culture):

Search for the target string "Å" in the string "xyza°".

Using the English (United States) - "en-US" culture:
Case sensitive:
  The string to search ends with the target string: False
Case insensitive:
  The string to search ends with the target string: True

Using the Swedish (Sweden) - "sv-SE" culture:
Case sensitive:
  The string to search ends with the target string: False
Case insensitive:
  The string to search ends with the target string: False

*/
```

Remarks

This method compares the `value` parameter to the substring at the end of this string that is the same length as `value`, and returns a value that indicates whether they are equal. To be equal, `value` must be a reference to this same instance or match the end of this string.

This method performs a word (culture-sensitive) comparison using the specified casing and culture.

See also

- InvariantCulture

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.EnumerateRunes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumeration of Rune from this string.

C#

```
public System.Text.StringRuneEnumerator EnumerateRunes ();
```

Returns

StringRuneEnumerator

A string rune enumerator.

Remarks

Invalid sequences are represented in the enumeration by Rune.ReplacementChar.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two String objects have the same value.

Overloads

Expand table

Equals(Object)	Determines whether this instance and a specified object, which must also be a String object, have the same value.
Equals(String)	Determines whether this instance and another specified String object have the same value.
Equals(String, String)	Determines whether two specified String objects have the same value.
Equals(String, StringComparison)	Determines whether this string and a specified String object have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.
Equals(String, String, StringComparison)	Determines whether two specified String objects have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.

Equals(Object)

Determines whether this instance and a specified object, which must also be a String object, have the same value.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The string to compare to this instance.

Returns

Boolean

`true` if `obj` is a String and its value is the same as this instance; otherwise, `false`. If `obj` is `null`, the method returns `false`.

Examples

The following example demonstrates the Equals method.

C#

```
// Sample for String.Equals(Object)
//            String.Equals(String)
//            String.Equals(String, String)
using System;
using System.Text;

class Sample1
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder("abcd");
        String str1 = "abcd";
        String str2 = null;
        Object o2 = null;

        Console.WriteLine();
        Console.WriteLine(" *  The value of String str1 is '{0}'.",
str1);
        Console.WriteLine(" *  The value of StringBuilder sb is '{0}'.",
sb.ToString());

        Console.WriteLine();
        Console.WriteLine("1a) String.Equals(Object). Object is a
StringBuilder, not a String.");
        Console.WriteLine("    Is str1 equal to sb?: {0}",
str1.Equals(sb));

        Console.WriteLine();
        Console.WriteLine("1b) String.Equals(Object). Object is a
String.");
        str2 = sb.ToString();
        o2 = str2;
        Console.WriteLine(" *  The value of Object o2 is '{0}'.", o2);
        Console.WriteLine("    Is str1 equal to o2?: {0}",
```

```
str1.Equals(o2));

        Console.WriteLine();
        Console.WriteLine(" 2) String.Equals(String)");
        Console.WriteLine(" *  The value of String str2 is '{0}'.",
str2);
        Console.WriteLine("    Is str1 equal to str2?: {0}",
str1.Equals(str2));

        Console.WriteLine();
        Console.WriteLine(" 3) String.Equals(String, String)");
        Console.WriteLine("    Is str1 equal to str2?: {0}",
String.Equals(str1, str2));
    }
}
/*
This example produces the following results:

 *  The value of String str1 is 'abcd'.
 *  The value of StringBuilder sb is 'abcd'.

1a) String.Equals(Object). Object is a StringBuilder, not a String.
    Is str1 equal to sb?: False

1b) String.Equals(Object). Object is a String.
 *  The value of Object o2 is 'abcd'.
    Is str1 equal to o2?: True

 2) String.Equals(String)
 *  The value of String str2 is 'abcd'.
    Is str1 equal to str2?: True

 3) String.Equals(String, String)
    Is str1 equal to str2?: True
*/
```

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

See also

- Object
- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(String)

Determines whether this instance and another specified String object have the same value.

```
C#

public bool Equals (string? value);
```

Parameters

`value` String

The string to compare to this instance.

Returns

Boolean

`true` if the value of the `value` parameter is the same as the value of this instance; otherwise, `false`. If `value` is `null`, the method returns `false`.

Implements

Equals(T)

Examples

The following example demonstrates the Equals method. It compares the title-cased word "File" with an equivalent word, its lowercase equivalent, its uppercase equivalent, and a word that contains LATIN SMALL LETTER DOTLESS I (U+0131)

instead of LATIN SMALL LETTER I (U+0069). Because the Equals(String) method performs an ordinal comparison, only the comparison with an identical word returns `true`.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        string word = "File";
        string[] others = { word.ToLower(), word, word.ToUpper(), "Fıle"
};
        foreach (string other in others)
        {
            if (word.Equals(other))
                Console.WriteLine("{0} = {1}", word, other);
            else
                Console.WriteLine("{0} {1} {2}", word, '\u2260', other);
        }
    }
}
// The example displays the following output:
//       File ≠ file
//       File = File
//       File ≠ FILE
//       File ≠ Fıle
```

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

See also

- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(String, String)

Determines whether two specified String objects have the same value.

```
C#

public static bool Equals (string? a, string? b);
```

Parameters

a String

The first string to compare, or `null`.

b String

The second string to compare, or `null`.

Returns

Boolean

`true` if the value of `a` is the same as the value of `b`; otherwise, `false`. If both `a` and `b` are `null`, the method returns `true`.

Examples

The following example demonstrates the Equals method.

```
C#

// Sample for String.Equals(Object)
//            String.Equals(String)
//            String.Equals(String, String)
using System;
using System.Text;
```

```
class Sample1
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder("abcd");
        String str1 = "abcd";
        String str2 = null;
        Object o2 = null;

        Console.WriteLine();
        Console.WriteLine(" *  The value of String str1 is '{0}'.",
str1);
        Console.WriteLine(" *  The value of StringBuilder sb is '{0}'.",
sb.ToString());

        Console.WriteLine();
        Console.WriteLine("1a) String.Equals(Object). Object is a
StringBuilder, not a String.");
        Console.WriteLine("    Is str1 equal to sb?: {0}",
str1.Equals(sb));

        Console.WriteLine();
        Console.WriteLine("1b) String.Equals(Object). Object is a
String.");
        str2 = sb.ToString();
        o2 = str2;
        Console.WriteLine(" *  The value of Object o2 is '{0}'.", o2);
        Console.WriteLine("    Is str1 equal to o2?: {0}",
str1.Equals(o2));

        Console.WriteLine();
        Console.WriteLine(" 2) String.Equals(String)");
        Console.WriteLine(" *  The value of String str2 is '{0}'.",
str2);
        Console.WriteLine("    Is str1 equal to str2?: {0}",
str1.Equals(str2));

        Console.WriteLine();
        Console.WriteLine(" 3) String.Equals(String, String)");
        Console.WriteLine("    Is str1 equal to str2?: {0}",
String.Equals(str1, str2));
    }
}
/*
This example produces the following results:

 *  The value of String str1 is 'abcd'.
 *  The value of StringBuilder sb is 'abcd'.

1a) String.Equals(Object). Object is a StringBuilder, not a String.
    Is str1 equal to sb?: False

1b) String.Equals(Object). Object is a String.
 *  The value of Object o2 is 'abcd'.
```

```
    Is str1 equal to o2?: True

 2) String.Equals(String)
 *  The value of String str2 is 'abcd'.
    Is str1 equal to str2?: True

 3) String.Equals(String, String)
    Is str1 equal to str2?: True
*/
```

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

See also

- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(String, StringComparison)

Determines whether this string and a specified String object have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.

C#

```
public bool Equals (string? value, StringComparison comparisonType);
```

Parameters

`value` String

The string to compare to this instance.

`comparisonType` StringComparison

One of the enumeration values that specifies how the strings will be compared.

Returns

Boolean

`true` if the value of the `value` parameter is the same as this string; otherwise, `false`.

Exceptions

ArgumentException

`comparisonType` is not a StringComparison value.

Examples

The following example creates a string array that consists of an uppercase "I", a lowercase "i", and a dotless "ı". It then calls the Equals(String, StringComparison) method to compare them by using each possible StringComparison enumeration value.

C#

```
using System;

class Sample
{
   public static void Main()
   {
      // Define a string array with the following three "I" characters:
      //      U+0069, U+0131, and U+0049.
      string[] threeIs = { "i", "ı", "I" };
      // Define Type object representing StringComparison type.
      Type scType = typeof(StringComparison);

      // Show the current culture (for culture-sensitive string
comparisons).
      Console.WriteLine("The current culture is {0}.\n",

System.Globalization.CultureInfo.CurrentCulture.Name);

      // Perform comparisons using each StringComparison member.
```

```
      foreach (string scName in Enum.GetNames(scType))
      {
         StringComparison sc = (StringComparison) Enum.Parse(scType,
scName);
         Console.WriteLine("Comparisons using {0}:", sc);
         // Compare each character in character array.
         for (int ctr = 0; ctr <= 1; ctr++)
         {
            string instanceChar = threeIs[ctr];
            for (int innerCtr = ctr + 1; innerCtr <=
threeIs.GetUpperBound(0); innerCtr++)
            {
               string otherChar = threeIs[innerCtr];
               Console.WriteLine("{0} (U+{1}) = {2} (U+{3}): {4}",
                                 instanceChar,
Convert.ToInt16(Char.Parse(instanceChar)).ToString("X4"),
                                 otherChar,
Convert.ToInt16(Char.Parse(otherChar)).ToString("X4"),
                                 instanceChar.Equals(otherChar, sc));
            }
            Console.WriteLine();
         }
      }
   }
}
// The example displays the following output:
//       The current culture is en-US.
//
//       Comparisons using CurrentCulture:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): False
//
//       ı (U+0131) = I (U+0049): False
//
//       Comparisons using CurrentCultureIgnoreCase:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): True
//
//       ı (U+0131) = I (U+0049): False
//
//       Comparisons using InvariantCulture:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): False
//
//       ı (U+0131) = I (U+0049): False
//
//       Comparisons using InvariantCultureIgnoreCase:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): True
//
//       ı (U+0131) = I (U+0049): False
//
//       Comparisons using Ordinal:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): False
```

```
//
//       ı (U+0131) = I (U+0049): False
//
//       Comparisons using OrdinalIgnoreCase:
//       i (U+0069) = ı (U+0131): False
//       i (U+0069) = I (U+0049): True
//
//       ı (U+0131) = I (U+0049): False
```

Remarks

The `comparisonType` parameter indicates whether the comparison should use the current or invariant culture, honor or ignore the case of the two strings being compared, or use word or ordinal sort rules.

See also

- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(String, String, StringComparison)

Determines whether two specified String objects have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.

C#

```
public static bool Equals (string? a, string? b, StringComparison
```

```
comparisonType);
```

Parameters

a String

The first string to compare, or `null`.

b String

The second string to compare, or `null`.

comparisonType StringComparison

One of the enumeration values that specifies the rules for the comparison.

Returns

Boolean

`true` if the value of the `a` parameter is equal to the value of the `b` parameter; otherwise, `false`.

Exceptions

ArgumentException

`comparisonType` is not a StringComparison value.

Examples

The following example compares three sets of strings by using each member of the StringComparison enumeration. The comparisons use the conventions of the English (United States), Thai (Thailand) and Turkish (Turkey) cultures. Note that the strings "a" and "a-" are considered equivalent in the "th-TH" culture but not in the others, while "i" and "İ" are considered equivalent in the "tr-TR" culture when case is ignored but not in the other cultures.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example3
{
    public static void Main()
```

```
    {
        String[] cultureNames = { "en-US", "th-TH", "tr-TR" };
        String[] strings1 = { "a", "i", "case", };
        String[] strings2 = { "a-", "\u0130", "Case" };
        StringComparison[] comparisons =
(StringComparison[])Enum.GetValues(typeof(StringComparison));

        foreach (var cultureName in cultureNames)
        {
            Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture(cultureName);
            Console.WriteLine("Current Culture: {0}",
CultureInfo.CurrentCulture.Name);
            for (int ctr = 0; ctr <= strings1.GetUpperBound(0); ctr++)
            {
                foreach (var comparison in comparisons)
                    Console.WriteLine("   {0} = {1} ({2}): {3}",
strings1[ctr],
                                      strings2[ctr], comparison,
                                      String.Equals(strings1[ctr],
strings2[ctr], comparison));

                Console.WriteLine();
            }
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//    Current Culture: en-US
//       a = a- (CurrentCulture): False
//       a = a- (CurrentCultureIgnoreCase): False
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): False
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
//
//
//    Current Culture: th-TH
//       a = a- (CurrentCulture): True
//       a = a- (CurrentCultureIgnoreCase): True
```

```
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): False
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
//
//
//    Current Culture: tr-TR
//       a = a- (CurrentCulture): False
//       a = a- (CurrentCultureIgnoreCase): False
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): True
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
```

Remarks

The `comparisonType` parameter indicates whether the comparison should use the current or invariant culture, honor or ignore the case of the two strings being compared, or use word or ordinal sort rules.

See also

- CompareOrdinal(String, String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Format Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of objects to strings based on the formats specified and inserts them into another string.

If you are new to the `String.Format` method, see Get started with the String.Format method for a quick overview.

Overloads

Expand table

Format(IFormatProvider, String, Object, Object, Object)	Replaces the format items in a string with the string representation of three specified objects. An parameter supplies culture-specific formatting information.
Format(String, Object, Object, Object)	Replaces the format items in a string with the string representation of three specified objects.
Format(IFormatProvider, String, Object, Object)	Replaces the format items in a string with the string representation of two specified objects. A parameter supplies culture-specific formatting information.
Format(String, Object, Object)	Replaces the format items in a string with the string representation of two specified objects.
Format(IFormatProvider, CompositeFormat, ReadOnlySpan<Object>)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format(IFormatProvider, String, Object)	Replaces the format item or items in a specified string with the string representation of the corresponding object. A parameter supplies culture-specific formatting information.
Format(IFormatProvider, String, Object[])	Replaces the format items in a string with the string representations of corresponding objects in a

	specified array. A parameter supplies culture-specific formatting information.
Format(String, Object[])	Replaces the format item in a specified string with the string representation of a corresponding object in a specified array.
Format(String, Object)	Replaces one or more format items in a string with the string representation of a specified object.
Format(IFormatProvider, CompositeFormat, Object[])	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format<TArg0,TArg1,TArg2>(IFormatProvider, CompositeFormat, TArg0, TArg1, TArg2)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format<TArg0,TArg1>(IFormatProvider, CompositeFormat, TArg0, TArg1)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.
Format<TArg0>(IFormatProvider, CompositeFormat, TArg0)	Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

Remarks

For more information about this API, see Supplemental API remarks for String.Format.

Format(IFormatProvider, String, Object, Object, Object)

Replaces the format items in a string with the string representation of three specified objects. An parameter supplies culture-specific formatting information.

C#

```
public static string Format (IFormatProvider? provider, string format,
object? arg0, object? arg1, object? arg2);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

`format` String

A composite format string.

`arg0` Object

The first object to format.

`arg1` Object

The second object to format.

`arg2` Object

The third object to format.

Returns

String

A copy of `format` in which the format items have been replaced by the string representations of `arg0`, `arg1`, and `arg2`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is less than zero, or greater than two.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result

string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert three expressions to their string representations and to embed those representations in a string. In performing the conversion, the method uses culture-sensitive formatting or a custom formatter. The method converts each Object argument to its string representation by calling its **ToString(IFormatProvider)** method or, if the object's corresponding format item includes a format string, by calling its **ToString(String,IFormatProvider)** method. If these methods don't exist, it calls the object's parameterless **ToString** method.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with an object that provides culture-sensitive or custom formatting and a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(String, Object, Object, Object)

Replaces the format items in a string with the string representation of three specified objects.

```
C#

public static string Format (string format, object? arg0, object? arg1,
object? arg2);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to format.

`arg1` Object

The second object to format.

`arg2` Object

The third object to format.

Returns

String

A copy of `format` in which the format items have been replaced by the string representations of `arg0`, `arg1`, and `arg2`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is less than zero, or greater than two.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert the value of three expressions to their string representations and to embed those representations in a string.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Example: Format three arguments

This example uses the Format(String, Object, Object, Object) method to create a string that illustrates the result of a Boolean `And` operation with two integer values. Note that the format string includes six format items, but the method has only three items in its parameter list, because each item is formatted in two different ways.

C#

```
string formatString = "    {0,10} ({0,8:X8})\n" +
                      "And {1,10} ({1,8:X8})\n" +
                      "  = {2,10} ({2,8:X8})";
int value1 = 16932;
int value2 = 15421;
string result = String.Format(formatString,
                              value1, value2, value1 & value2);
Console.WriteLine(result);
```

```
// The example displays the following output:
//                16932 (00004224)
//       And      15421 (00003C3D)
//         =         36 (00000024)
```

See also

- Formatting Types in .NET
- Composite Formatting

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(IFormatProvider, String, Object, Object)

Replaces the format items in a string with the string representation of two specified objects. A parameter supplies culture-specific formatting information.

C#

```
public static string Format (IFormatProvider? provider, string format,
object? arg0, object? arg1);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

format String

A composite format string.

`arg0` Object

The first object to format.

`arg1` Object

The second object to format.

Returns

String

A copy of `format` in which format items are replaced by the string representations of `arg0` and `arg1`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is not zero or one.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert two expressions to their string representations and to embed those representations in a string. In performing the conversion, the method uses culture-sensitive formatting or a custom formatter. The method converts each Object argument to its string representation by calling its **ToString(IFormatProvider)** method or, if the object's

corresponding format item includes a format string, by calling its **ToString(String,IFormatProvider)** method. If these methods don't exist, it calls the object's parameterless **ToString** method.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with an object that provides culture-sensitive or custom formatting and a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(String, Object, Object)

Replaces the format items in a string with the string representation of two specified objects.

C#

```
public static string Format (string format, object? arg0, object? arg1);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The first object to format.

`arg1` Object

The second object to format.

Returns

String

A copy of `format` in which format items are replaced by the string representations of `arg0` and `arg1`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is not zero or one.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert the value of two expressions to their string representations and to embed those representations in a string.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Example: Format two arguments

This example uses the Format(String, Object, Object) method to display time and temperature data stored in a generic Dictionary<TKey,TValue> object. Note that the format string has three format items, although there are only two objects to format. This is because the first object in the list (a date and time value) is used by two format items: The first format item displays the time, and the second displays the date.

C#

```
Dictionary<DateTime, Double> temperatureInfo = new Dictionary<DateTime,
Double>();
temperatureInfo.Add(new DateTime(2010, 6, 1, 14, 0, 0), 87.46);
temperatureInfo.Add(new DateTime(2010, 12, 1, 10, 0, 0), 36.81);

Console.WriteLine("Temperature Information:\n");
string output;
foreach (var item in temperatureInfo)
{
   output = String.Format("Temperature at {0,8:t} on {0,9:d}:
{1,5:N1}°F",
                          item.Key, item.Value);
   Console.WriteLine(output);
}
// The example displays output like the following:
//       Temperature Information:
//
```

```
//       Temperature at  2:00 PM on  6/1/2010:  87.5°F
//       Temperature at 10:00 AM on 12/1/2010:  36.8°F
```

See also

- Formatting Types in .NET
- Composite Formatting
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(IFormatProvider, CompositeFormat, ReadOnlySpan<Object>)

Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

C#

```
public static string Format (IFormatProvider? provider,
System.Text.CompositeFormat format, ReadOnlySpan<object?> args);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`args` ReadOnlySpan<Object>

A span of objects to format.

Returns

String

The formatted string.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Format(IFormatProvider, String, Object)

Replaces the format item or items in a specified string with the string representation of the corresponding object. A parameter supplies culture-specific formatting information.

```
C#
```

```
public static string Format (IFormatProvider? provider, string format,
object? arg0);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` String

A composite format string.

`arg0` Object

The object to format.

Returns

String

A copy of `format` in which the format item or items have been replaced by the string representation of `arg0`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is not zero.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result

string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert the value of an expression to its string representation and to embed that representation in a string. In performing the conversion, the method uses culture-sensitive formatting or a custom formatter. The method converts `arg0` to its string representation by calling its **ToString(IFormatProvider)** method or, if the object's corresponding format item includes a format string, by calling its **ToString(String,IFormatProvider)** method. If these methods don't exist, it calls the object's parameterless **ToString** method.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with an object that provides culture-sensitive or custom formatting and a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(IFormatProvider, String, Object[])

Replaces the format items in a string with the string representations of corresponding objects in a specified array. A parameter supplies culture-specific formatting information.

```
C#

public static string Format (IFormatProvider? provider, string format,
params object?[] args);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` String

A composite format string.

`args` Object[]

An object array that contains zero or more objects to format.

Returns

String

A copy of `format` in which the format items have been replaced by the string representation of the corresponding objects in `args`.

Exceptions

ArgumentNullException

`format` or `args` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is less than zero, or greater than or equal to the length of the `args` array.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert four or more expressions to their string representations and to embed those representations in a string. In performing the conversion, the method uses culture-sensitive formatting or a custom formatter. The method converts each Object argument to its string representation by calling its **ToString(IFormatProvider)** method or, if the object's corresponding format item includes a format string, by calling its **ToString(String,IFormatProvider)** method. If these methods don't exist, it calls the object's parameterless **ToString** method.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with an object that provides culture-sensitive or custom formatting and a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Example: Culture-sensitive formatting

This example uses the Format(IFormatProvider, String, Object[]) method to display the string representation of some date and time values and numeric values by using several different cultures.

C#

```
string[] cultureNames = { "en-US", "fr-FR", "de-DE", "es-ES" };

DateTime dateToDisplay = new DateTime(2009, 9, 1, 18, 32, 0);
```

```
double value = 9164.32;

Console.WriteLine("Culture     Date
Value\n");
foreach (string cultureName in cultureNames)
{
   System.Globalization.CultureInfo culture = new
System.Globalization.CultureInfo(cultureName);
   string output = String.Format(culture, "{0,-11} {1,-35:D} {2:N}",
                                 culture.Name, dateToDisplay, value);
   Console.WriteLine(output);
}
// The example displays the following output:
//    Culture     Date                                Value
//
//    en-US       Tuesday, September 01, 2009         9,164.32
//    fr-FR       mardi 1 septembre 2009              9 164,32
//    de-DE       Dienstag, 1. September 2009         9.164,32
//    es-ES       martes, 01 de septiembre de 2009    9.164,32
```

See also

- DateTimeFormatInfo
- ICustomFormatter
- IFormatProvider
- NumberFormatInfo
- Formatting Types in .NET
- Composite Formatting
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(String, Object[])

Replaces the format item in a specified string with the string representation of a corresponding object in a specified array.

C#

```
public static string Format (string format, params object?[] args);
```

Parameters

`format` String

A composite format string.

`args` Object[]

An object array that contains zero or more objects to format.

Returns

String

A copy of `format` in which the format items have been replaced by the string representation of the corresponding objects in `args`.

Exceptions

ArgumentNullException

`format` or `args` is `null`.

FormatException

`format` is invalid.

-or-

The index of a format item is less than zero, or greater than or equal to the length of the `args` array.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert the value of four or more expressions to their string representations and to embed those representations in a string. Since the `args` parameter is marked with the System.ParamArrayAttribute attribute, you can pass the objects to the method as individual arguments or as an Object array.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Example: Format more than three arguments

This example creates a string that contains data on the high and low temperature on a particular date. The composite format string has five format items in the C# example and six in the Visual Basic example. Two of the format items define the width of their corresponding value's string representation, and the first format item also includes a standard date and time format string.

C#

```
DateTime date1 = new DateTime(2009, 7, 1);
TimeSpan hiTime = new TimeSpan(14, 17, 32);
decimal hiTemp = 62.1m;
TimeSpan loTime = new TimeSpan(3, 16, 10);
decimal loTemp = 54.8m;

string result1 = String.Format("Temperature on {0:d}:\n{1,11}: {2}
degrees (hi)\n{3,11}: {4} degrees (lo)",
                               date1, hiTime, hiTemp, loTime, loTemp);
Console.WriteLine(result1);
Console.WriteLine();

string result2 = String.Format("Temperature on {0:d}:\n{1,11}: {2}
degrees (hi)\n{3,11}: {4} degrees (lo)",
                               new object[] { date1, hiTime, hiTemp,
loTime, loTemp });
Console.WriteLine(result2);
// The example displays output like the following:
//       Temperature on 7/1/2009:
//          14:17:32: 62.1 degrees (hi)
//          03:16:10: 54.8 degrees (lo)
//       Temperature on 7/1/2009:
//          14:17:32: 62.1 degrees (hi)
//          03:16:10: 54.8 degrees (lo)
```

You can also pass the objects to be formatted as an array rather than as an argument list.

C#

```
using System;

public class CityInfo
{
   public CityInfo(String name, int population, Decimal area, int year)
   {
      this.Name = name;
      this.Population = population;
      this.Area = area;
      this.Year = year;
   }

   public readonly String Name;
   public readonly int Population;
   public readonly Decimal Area;
   public readonly int Year;
}

public class Example
{
```

```
    public static void Main()
    {
        CityInfo nyc2010 = new CityInfo("New York", 8175133, 302.64m,
2010);
        ShowPopulationData(nyc2010);
        CityInfo sea2010 = new CityInfo("Seattle", 608660, 83.94m, 2010);
        ShowPopulationData(sea2010);
    }

    private static void ShowPopulationData(CityInfo city)
    {
        object[] args = { city.Name, city.Year, city.Population, city.Area
};
        String result = String.Format("{0} in {1}: Population {2:N0}, Area
{3:N1} sq. feet",
                                      args);
        Console.WriteLine(result);
    }
}
// The example displays the following output:
//       New York in 2010: Population 8,175,133, Area 302.6 sq. feet
//       Seattle in 2010: Population 608,660, Area 83.9 sq. feet
```

See also

- Formatting Types in .NET
- Composite Formatting
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Format(String, Object)

Replaces one or more format items in a string with the string representation of a specified object.

```
C#
public static string Format (string format, object? arg0);
```

Parameters

`format` String

A composite format string.

`arg0` Object

The object to format.

Returns

String

A copy of `format` in which any format items are replaced by the string representation of `arg0`.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The format item in `format` is invalid.

-or-

The index of a format item is not zero.

Remarks

> **Important**
>
> Instead of calling the **String.Format** method or using **composite format strings**, you can use *interpolated strings* if your language supports them. An interpolated string is a string that contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see **String interpolation (C# Reference)** and **Interpolated Strings (Visual Basic Reference)**.

This method uses the composite formatting feature to convert the value of an expression to its string representation and to embed that representation in a string.

However, when calling the `String.Format` method, it's not necessary to focus on the particular overload that you want to call. Instead, you can call the method with a composite format string that includes one or more format items. You assign each format item a numeric index; the first index starts at 0. In addition to the initial string, your method call should have as many additional arguments as it has index values. For example, a string whose format items have indexes of 0 and 1 should have 2 arguments; one with indexes 0 through 5 should have 6 arguments. The language compiler will then resolve your method call to a particular overload of the `String.Format` method.

For more detailed documentation on using the `String.Format` method, see Get started with the String.Format method and Which method do I call?.

Example: Formatting a single argument

The following example uses the Format(String, Object) method to embed an individual's age in the middle of a string.

C#

```
DateTime birthdate = new DateTime(1993, 7, 28);
DateTime[] dates = { new DateTime(1993, 8, 16),
                     new DateTime(1994, 7, 28),
                     new DateTime(2000, 10, 16),
                     new DateTime(2003, 7, 27),
                     new DateTime(2007, 5, 27) };

foreach (DateTime dateValue in dates)
{
```

```
    TimeSpan interval = dateValue - birthdate;
    // Get the approximate number of years, without accounting for leap
 years.
    int years = ((int) interval.TotalDays) / 365;
    // See if adding the number of years exceeds dateValue.
    string output;
    if (birthdate.AddYears(years) <= dateValue) {
       output = String.Format("You are now {0} years old.", years);
       Console.WriteLine(output);
    }
    else {
       output = String.Format("You are now {0} years old.", years - 1);
       Console.WriteLine(output);
    }
 }
 // The example displays the following output:
 //       You are now 0 years old.
 //       You are now 1 years old.
 //       You are now 7 years old.
 //       You are now 9 years old.
 //       You are now 13 years old.
```

See also

- Formatting Types in .NET
- Composite Formatting
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings
- Enumeration Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Format(IFormatProvider, CompositeFormat, Object[])

Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

C#

```
public static string Format (IFormatProvider? provider,
System.Text.CompositeFormat format, params object?[] args);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`args` Object[]

An array of objects to format.

Returns

String

The formatted string.

Exceptions

ArgumentNullException

`format` or `args` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Format<TArg0,TArg1,TArg2>(IFormatProvider, CompositeFormat, TArg0, TArg1, TArg2)

Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

C#

```
public static string Format<TArg0,TArg1,TArg2> (IFormatProvider?
provider, System.Text.CompositeFormat format, TArg0 arg0, TArg1 arg1,
TArg2 arg2);
```

Type Parameters

`TArg0`

The type of the first object to format.

`TArg1`

The type of the second object to format.

`TArg2`

The type of the third object to format.

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`arg0` TArg0

The first object to format.

`arg1` TArg1

The second object to format.

`arg2` TArg2

The third object to format.

Returns

String

The formatted string.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Format<TArg0,TArg1>(IFormatProvider, CompositeFormat, TArg0, TArg1)

Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

C#

```
public static string Format<TArg0,TArg1> (IFormatProvider? provider,
System.Text.CompositeFormat format, TArg0 arg0, TArg1 arg1);
```

Type Parameters

`TArg0`

The type of the first object to format.

`TArg1`

The type of the second object to format.

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`arg0` TArg0

The first object to format.

`arg1` TArg1

The second object to format.

Returns

String

The formatted string.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Format<TArg0>(IFormatProvider, CompositeFormat, TArg0)

Replaces the format item or items in a CompositeFormat with the string representation of the corresponding objects in the specified format.

C#

```
public static string Format<TArg0> (IFormatProvider? provider,
System.Text.CompositeFormat format, TArg0 arg0);
```

Type Parameters

`TArg0`

The type of the first object to format.

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

`format` CompositeFormat

A CompositeFormat.

`arg0` TArg0

The first object to format.

Returns

String

The formatted string.

Exceptions

ArgumentNullException

`format` is `null`.

FormatException

The index of a format item is greater than or equal to the number of supplied arguments.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves an object that can iterate through the individual characters in this string.

C#

```
public CharEnumerator GetEnumerator ();
```

Returns

CharEnumerator

An enumerator object.

Examples

The following example iterates the characters in several strings and displays information about their individual characters. It uses the language iteration construct rather than a call to the GetEnumerator method.

C#

```
using System;

class Example
{
   public static void Main()
   {
      EnumerateAndDisplay("Test Case");
      EnumerateAndDisplay("This is a sentence.");
      EnumerateAndDisplay("Has\ttwo\ttabs" );
      EnumerateAndDisplay("Two\nnew\nlines");
   }

   static void EnumerateAndDisplay(String phrase)
   {
      Console.WriteLine("The characters in the string \"{0}\" are:",
                        phrase);
```

```
      int CharCount = 0;
      int controlChars = 0;
      int alphanumeric = 0;
      int punctuation = 0;

      foreach (var ch in phrase) {
         Console.Write(Char.IsControl(ch) ? $"{ch}" : $"0x{(ushort)ch:X4}");

         if (Char.IsLetterOrDigit(ch))
            alphanumeric++;
         else if (Char.IsControl(ch))
            controlChars++;
         else if (Char.IsPunctuation(ch))
            punctuation++;
         CharCount++;
      }

      Console.WriteLine("\n   Total characters:        {0,3}", CharCount);
      Console.WriteLine("   Alphanumeric characters: {0,3}", alphanumeric);
      Console.WriteLine("   Punctuation characters:  {0,3}", punctuation);
      Console.WriteLine("   Control Characters:      {0,3}\n",
controlChars);
   }
}
// The example displays the following output:
//    The characters in the string "Test Case" are:
//    'T' 'e' 's' 't' ' ' 'C' 'a' 's' 'e'
//       Total characters:          9
//       Alphanumeric characters:   8
//       Punctuation characters:    0
//       Control Characters:        0
//
//    The characters in the string "This is a sentence." are:
//    'T' 'h' 'i' 's' ' ' 'i' 's' ' ' 'a' ' ' 's' 'e' 'n' 't' 'e' 'n' 'c'
'e' '.'
//       Total characters:         19
//       Alphanumeric characters:  15
//       Punctuation characters:    1
//       Control Characters:        0
//
//    The characters in the string "Has       two     tabs" are:
//    'H' 'a' 's' '0x0009' 't' 'w' 'o' '0x0009' 't' 'a' 'b' 's'
//       Total characters:         12
//       Alphanumeric characters:  10
//       Punctuation characters:    0
//       Control Characters:        2
//
//    The characters in the string "Two
//    new
//    lines" are:
//    'T' 'w' 'o' '0x000A' 'n' 'e' 'w' '0x000A' 'l' 'i' 'n' 'e' 's'
//       Total characters:         13
//       Alphanumeric characters:  11
```

```
//       Punctuation characters:    0
//       Control Characters:        2
```

Remarks

> **Tip**
>
> Rather than calling the **GetEnumerator** method to retrieve a **CharEnumerator** object that you then use to enumerate a string, you should instead use your language's iteration construct (in C#, in F#, in C++/CLR, and in Visual Basic). **foreach** in C#, **for..in** in F#, **for each** in C++/CLR, and **For Each** in Visual Basic).

This method enables you to iterate the individual characters in a string. For example, the Visual Basic `For Each`, the F# `for..in` expression, and C# `foreach` statements invoke this method to return a CharEnumerator object that can provide read-only access to the characters in this string instance.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IEnumerator
- IEnumerable
- Chars[Int32]

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

String.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetHashCode(ReadOnlySpan<Char>, StringComparison)	Returns the hash code for the provided read-only character span using the specified rules.
GetHashCode(StringComparison)	Returns the hash code for this string using the specified rules.
GetHashCode()	Returns the hash code for this string.
GetHashCode(ReadOnlySpan<Char>)	Returns the hash code for the provided read-only character span.

GetHashCode(ReadOnlySpan<Char>, StringComparison)

Returns the hash code for the provided read-only character span using the specified rules.

C#

```
public static int GetHashCode (ReadOnlySpan<char> value, StringComparison
comparisonType);
```

Parameters

`value` ReadOnlySpan<Char>

A read-only character span.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Int32

A 32-bit signed integer hash code.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

GetHashCode(StringComparison)

Returns the hash code for this string using the specified rules.

C#

```
public int GetHashCode (StringComparison comparisonType);
```

Parameters

`comparisonType` StringComparison

One of the enumeration values that specifies the rules to use in the comparison.

Returns

Int32

A 32-bit signed integer hash code.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

GetHashCode()

Returns the hash code for this string.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Examples

The following example demonstrates the GetHashCode method using various input strings.

C#

```
using System;

class GetHashCode
{
    public static void Main()
    {
        DisplayHashCode( "" );
        DisplayHashCode( "a" );
        DisplayHashCode( "ab" );
        DisplayHashCode( "abc" );
        DisplayHashCode( "abd" );
        DisplayHashCode( "abe" );
        DisplayHashCode( "abcdef" );
        DisplayHashCode( "abcdeg" );
        DisplayHashCode( "abcdeh" );
        DisplayHashCode( "abcdei" );
        DisplayHashCode( "Abcdeg" );
        DisplayHashCode( "Abcdeh" );
        DisplayHashCode( "Abcdei" );
    }

    static void DisplayHashCode( String Operand )
```

```
    {
        int     HashCode = Operand.GetHashCode( );
        Console.WriteLine("The hash code for \"{0}\" is: 0x{1:X8}, {1}",
                          Operand, HashCode );
    }
}
/*
      This example displays output like the following:
      The hash code for "" is: 0x2D2816FE, 757602046
      The hash code for "a" is: 0xCDCAB7BF, -842352705
      The hash code for "ab" is: 0xCDE8B7BF, -840386625
      The hash code for "abc" is: 0x2001D81A, 536991770
      The hash code for "abd" is: 0xC2A94CB5, -1029092171
      The hash code for "abe" is: 0x6550C150, 1699791184
      The hash code for "abcdef" is: 0x1762906D, 392335469
      The hash code for "abcdeg" is: 0x1763906D, 392401005
      The hash code for "abcdeh" is: 0x175C906D, 391942253
      The hash code for "abcdei" is: 0x175D906D, 392007789
      The hash code for "Abcdeg" is: 0x1763954D, 392402253
      The hash code for "Abcdeh" is: 0x175C954D, 391943501
      The hash code for "Abcdei" is: 0x175D954D, 392009037
*/
```

Remarks

The behavior of GetHashCode is dependent on its implementation, which might change from one version of the common language runtime to another. A reason why this might happen is to improve the performance of GetHashCode.

> **Important**
>
> If two string objects are equal, the **GetHashCode** method returns identical values. However, there is not a unique hash code value for each unique string value. Different strings can return the same hash code.
>
> The hash code itself is not guaranteed to be stable. Hash codes for identical strings can differ across .NET implementations, across .NET versions, and across .NET platforms (such as 32-bit and 64-bit) for a single version of .NET. In some cases, they can even differ by application domain. This implies that two subsequent runs of the same program may return different hash codes.
>
> As a result, hash codes should never be used outside of the application domain in which they were created, they should never be used as key fields in a collection, and they should never be persisted.
>
> Finally, don't use the hash code instead of a value returned by a cryptographic hashing function if you need a cryptographically strong hash. For cryptographic

hashes, use a class derived from the **System.Security.Cryptography.HashAlgorithm** or **System.Security.Cryptography.KeyedHashAlgorithm** class.

For more information about hash codes, see **Object.GetHashCode**.

In .NET Framework desktop apps, you can use the <UseRandomizedStringHashAlgorithm> element to generate unique hash codes on a per-application domain basis. This can reduce the number of collisions and improve the overall performance of insertions and lookups that use hash tables. The following example shows how to use the <UseRandomizedStringHashAlgorithm> element. It defines a `DisplayString` class that includes a private string constant, `s`, whose value is "This is a string." It also includes a `ShowStringHashCode` method that displays the string value and its hash code along with the name of the application domain in which the method is executing.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Show hash code in current domain.
      DisplayString display = new DisplayString();
      display.ShowStringHashCode();

      // Create a new app domain and show string hash code.
      AppDomain domain = AppDomain.CreateDomain("NewDomain");
      var display2 = (DisplayString)
domain.CreateInstanceAndUnwrap(typeof(Example).Assembly.FullName,

"DisplayString");
      display2.ShowStringHashCode();
   }
}

public class DisplayString : MarshalByRefObject
{
   private String s = "This is a string.";

   public override bool Equals(Object obj)
   {
      String s2 = obj as String;
      if (s2 == null)
         return false;
      else
         return s == s2;
```

```
    }

    public bool Equals(String str)
    {
       return s == str;
    }

    public override int GetHashCode()
    {
       return s.GetHashCode();
    }

    public override String ToString()
    {
       return s;
    }

    public void ShowStringHashCode()
    {
       Console.WriteLine("String '{0}' in domain '{1}': {2:X8}",
                         s, AppDomain.CurrentDomain.FriendlyName,
                         s.GetHashCode());
    }
 }
```

When you run the example without supplying a configuration file, it displays output similar to the following. Note that the hash codes for the string are identical in the two application domains.

```
String 'This is a string.' in domain 'PerDomain.exe': 941BCEAC
String 'This is a string.' in domain 'NewDomain': 941BCEAC
```

However, if you add the following configuration file to the example's directory and then run the example, the hash codes for the same string will differ by application domain.

XML

```
<?xml version ="1.0"?>
<configuration>
   <runtime>
      <UseRandomizedStringHashAlgorithm enabled="1" />
   </runtime>
</configuration>
```

When the configuration file is present, the example displays the following output:

```
String 'This is a string.' in domain 'PerDomain.exe': 5435776D
String 'This is a string.' in domain 'NewDomain': 75CC8236
```

> **Important**
>
> Hash codes are used to insert and retrieve keyed objects from hash tables efficiently. However, hash codes don't uniquely identify strings. Identical strings have equal hash codes, but the common language runtime can also assign the same hash code to different strings. In addition, hash codes can vary by version of .NET, by platform within a single version, and by application domain. Because of this, you should not serialize or persist hash code values, nor should you use them as keys in a hash table or dictionary.

For additional information about the use of hash codes and the `GetHashCode` method, see Object.GetHashCode.

Notes to Callers

The value returned by GetHashCode() is platform-dependent. It differs on the 32-bit and 64-bit versions of .NET Framework. It also can differ between versions of .NET Framework and .NET Core.

See also

- GetHashCode()
- <UseRandomizedStringHashAlgorithm> Element

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

GetHashCode(ReadOnlySpan<Char>)

Returns the hash code for the provided read-only character span.

```
C#
public static int GetHashCode (ReadOnlySpan<char> value);
```

Parameters

`value` ReadOnlySpan<Char>

A read-only character span.

Returns

Int32

A 32-bit signed integer hash code.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.GetPinnableReference Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a reference to the element of the string at index zero.

This method is intended to support .NET compilers and is not intended to be called by user code.

```
C#
public ref readonly char GetPinnableReference ();
```

Returns

Char

A reference to the first character in the string, or a reference to the string's null terminator if the string is empty.

Exceptions

NullReferenceException

The string is null.

Remarks

The `GetPinnableReference` method returns a character that can be used for pinning a String in memory. It is required to support the use of a String within a fixed statement.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for the String class.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, String.

Implements

GetTypeCode()

Examples

The following example displays the TypeCode enumerated constant for the String type.

C#

```
// Sample for String.GetTypeCode()
using System;

class Sample
{
    public static void Main()
    {
    String str = "abc";
    TypeCode tc = str.GetTypeCode();
    Console.WriteLine("The type code for '{0}' is {1}, which represents
{2}.",
                          str, tc.ToString("D"), tc.ToString("F"));
    }
}
```

```
/*
This example produces the following results:
The type code for 'abc' is 18, which represents String.
*/
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports the zero-based index of the first occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.

Overloads

Expand table

IndexOf(String, Int32, Int32)	Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position and examines a specified number of character positions.
IndexOf(String, Int32, Int32, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string, the number of characters in the current string to search, and the type of search to use for the specified string.
IndexOf(String, Int32, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string and the type of search to use for the specified string.
IndexOf(String, StringComparison)	Reports the zero-based index of the first occurrence of the specified string in the current String object. A parameter specifies the type of search to use for the specified string.
IndexOf(Char, Int32, Int32)	Reports the zero-based index of the first occurrence of the specified character in this instance. The search starts at a specified character position and examines a specified number of character positions.
IndexOf(Char, StringComparison)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string. A parameter specifies the type of search to use for the specified character.
IndexOf(Char, Int32)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string. The search starts at a specified

	character position.
IndexOf(String)	Reports the zero-based index of the first occurrence of the specified string in this instance.
IndexOf(Char)	Reports the zero-based index of the first occurrence of the specified Unicode character in this string.
IndexOf(String, Int32)	Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position.

IndexOf(String, Int32, Int32)

Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position and examines a specified number of character positions.

C#

```
public int IndexOf (string value, int startIndex, int count);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position.

`count` Int32

The number of character positions to examine.

Returns

Int32

The zero-based index position of `value` from the start of the current instance if that string is found, or -1 if it is not. If `value` is Empty, the return value is `startIndex`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`count` or `startIndex` is negative.

-or-

`startIndex` is greater than the length of this string.

-or-

`count` is greater than the length of this string minus `startIndex`.

Examples

The following example finds the index of all occurrences of the string "he" within a substring of another string. Note that the number of characters to be searched must be recalculated for each search iteration.

C#

```
string br1 = "0----+----1----+----2----+----3----+----4----+----5----+---
-6----+---";
string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456789012345678";
string str = "Now is the time for all good men to come to the aid of
their country.";
int start;
int at;
int end;
int count;

end = str.Length;
start = end/2;
Console.WriteLine();
Console.WriteLine("All occurrences of 'he' from position {0} to {1}.",
start, end-1);
Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1, br2,
str);
Console.Write("The string 'he' occurs at position(s): ");

count = 0;
at = 0;
while((start <= end) && (at > -1))
{
    // start+count must be a position within -str-.
    count = end - start;
    at = str.IndexOf("he", start, count);
    if (at == -1) break;
    Console.Write("{0} ", at);
```

```
    start = at+1;
}
Console.WriteLine();

/*
This example produces the following results:

All occurrences of 'he' from position 34 to 68.
0----+----1----+----2----+----3----+----4----+----5----+----6----+---
0123456789012345678901234567890123456789012345678901234567890123456789012345678
Now is the time for all good men to come to the aid of their country.

The string 'he' occurs at position(s): 45 56

*/
```

Remarks

Index numbering starts from 0 (zero). The `startIndex` parameter can range from 0 to the length of the string instance.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture. The search begins at `startIndex` and continues to `startIndex` + `count` -1. The character at `startIndex` + `count` is not included in the search.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String, Int32, Int32) method always returns `startIndex`, which is the character position at which the search begins. In the following example, the IndexOf(String, Int32, Int32) method is used to find the position of a soft hyphen (U+00AD) followed by an "m" starting in the third through sixth character positions in two strings. Only one of the strings contains the required substring. If the example is run on the .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string when it performs a culture-sensitive comparison. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method fails to return the index of the soft hyphen but instead returns the index of the "m".

C#

```
using System;
```

```
public class Example
{
    public static void Main()
    {
        string searchString = "\u00ADm";
        string s1 = "ani\u00ADmal" ;
        string s2 = "animal";

        Console.WriteLine(s1.IndexOf(searchString, 2, 4));
        Console.WriteLine(s2.IndexOf(searchString, 2, 4));

        // The example displays the following output:
        //       4
        //       3
    }
}
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To use the comparison rules of the current culture to perform this operation, signal your intention explicitly by calling the IndexOf(String, Int32, Int32, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32
- CultureInfo
- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(String, Int32, Int32, StringComparison)

Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string, the number of characters in the current string to search, and the type of search to use for the specified string.

C#

```
public int IndexOf (string value, int startIndex, int count,
StringComparison comparisonType);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position.

`count` Int32

The number of character positions to examine.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The zero-based index position of the `value` parameter from the start of the current instance if that string is found, or -1 if it is not. If `value` is Empty, the return value is `startIndex`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`count` or `startIndex` is negative.

-or-

`startIndex` is greater than the length of this instance.

-or-

`count` is greater than the length of this string minus `startIndex`.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the IndexOf method that find the first occurrence of a string within another string using different values of the StringComparison enumeration.

C#

```
// This code example demonstrates the
// System.String.IndexOf(String, ..., StringComparison) methods.

using System;
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the first occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
```

```
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";

    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
                      Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                      CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
// the string was not found.
// Search using different values of StringComparison. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, cat.Length, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, sc);
```

```
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the first occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire ca°t"

Part 1: Start index and count are specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

*/
```

Remarks

Index numbering starts from 0 (zero). The `startIndex` parameter can range from 0 to the length of the string instance.

The search begins at `startIndex` and continues to `startIndex` + `count` -1. The character at `startIndex` + `count` is not included in the search.

The `comparisonType` parameter specifies to search for the `value` parameter using the current or invariant culture, using a case-sensitive or case-insensitive search, and using word or ordinal comparison rules.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `comparisonType` is not Ordinal or OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String, Int32, Int32, StringComparison) method always returns `startIndex`, which is the character position at which the search begins.

In the following example, the IndexOf(String, Int32, Int32, StringComparison) method is used to find the position of a soft hyphen (U+00AD) followed by an "m" starting in the third through sixth character positions in two strings. Only one of the strings contains the required substring. If the example is run on the .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string when it performs a culture-sensitive comparison. When it performs an ordinal comparison, however, it finds the substring only in the first string. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method fails to return the index of the soft hyphen but instead returns the index of the "m" when it performs a culture-sensitive comparison. The method returns the index of the soft hyphen in the first string only when it performs an ordinal comparison.

C#

```
using System;

public class Example
{
    public static void Main()
    {
```

```
        string searchString = "\u00ADm";
        string s1 = "ani\u00ADmal" ;
        string s2 = "animal";

        Console.WriteLine(s1.IndexOf(searchString, 2, 4,
StringComparison.CurrentCulture));
        Console.WriteLine(s1.IndexOf(searchString, 2, 4,
StringComparison.Ordinal));
        Console.WriteLine(s2.IndexOf(searchString, 2, 4,
StringComparison.CurrentCulture));
        Console.WriteLine(s2.IndexOf(searchString, 2, 4,
StringComparison.Ordinal));

        // The example displays the following output:
        //       4
        //       3
        //       3
        //       -1
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(String, Int32, StringComparison)

Reports the zero-based index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string and the type of search to use for the specified string.

C#

```
public int IndexOf (string value, int startIndex, StringComparison
comparisonType);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The zero-based index position of the `value` parameter from the start of the current instance if that string is found, or -1 if it is not. If `value` is Empty, the return value is `startIndex`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than 0 (zero) or greater than the length of this string.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the IndexOf method that find the first occurrence of a string within another string using different values of the StringComparison enumeration.

C#

```
// This code example demonstrates the
// System.String.IndexOf(String, ..., StringComparison) methods.

using System;
using System.Threading;
```

```
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the first occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";

    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
                      Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                      CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
// the string was not found.
```

```
// Search using different values of StringComparison. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, cat.Length, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the first occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire ca°t"

Part 1: Start index and count are specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
```

```
Comparison: OrdinalIgnoreCase            Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

*/
```

Remarks

Index numbering starts from 0. The `startIndex` parameter can range from 0 to the length of the string instance. If `startIndex` equals the length of the string instance, the method returns -1.

The `comparisonType` parameter specifies to search for the `value` parameter using the current or invariant culture, using a case-sensitive or case-insensitive search, and using word or ordinal comparison rules.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `comparisonType` is not Ordinal or OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String, Int32, StringComparison) method always returns `startIndex`, which is the character position at which the search begins.

In the following example, the IndexOf(String, Int32, StringComparison) method is used to find the position of a soft hyphen (U+00AD) followed by an "m" starting with the third character position in two strings. Only one of the strings contains the required substring. If the example is run on the .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string when it performs a culture-sensitive comparison. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method fails to return the index of the soft hyphen but instead returns the index of the "m". The method returns the index of the soft hyphen in the first string only when it performs an ordinal comparison.

C#

```
using System;

public class Example
{
    public static void Main()
    {

        string searchString = "\u00ADm";
        string s1 = "ani\u00ADmal" ;
        string s2 = "animal";

        Console.WriteLine(s1.IndexOf(searchString, 2,
StringComparison.CurrentCulture));
        Console.WriteLine(s1.IndexOf(searchString, 2,
StringComparison.Ordinal));
        Console.WriteLine(s2.IndexOf(searchString, 2,
StringComparison.CurrentCulture));
        Console.WriteLine(s2.IndexOf(searchString, 2,
StringComparison.Ordinal));

        // The example displays the following output:
        //       4
        //       3
        //       3
        //       -1
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(String, StringComparison)

Reports the zero-based index of the first occurrence of the specified string in the current String object. A parameter specifies the type of search to use for the specified string.

```
C#

public int IndexOf (string value, StringComparison comparisonType);
```

Parameters

`value` String

The string to seek.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The index position of the `value` parameter if that string is found, or -1 if it is not. If `value` is Empty, the return value is 0.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the IndexOf method that find the first occurrence of a string within another string using different values of the StringComparison enumeration.

```
C#

// This code example demonstrates the
// System.String.IndexOf(String, ..., StringComparison) methods.

using System;
```

```
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the first occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";

    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
                      Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                      CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
```

```
// the string was not found.
// Search using different values of StringComparison. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, cat.Length, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, 0, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparison.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.IndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the first occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire caºt"

Part 1: Start index and count are specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
```

```
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

*/
```

Remarks

Index numbering starts from zero.

The `comparisonType` parameter specifies to search for the `value` parameter using the current or invariant culture, using a case-sensitive or case-insensitive search, and using word or ordinal comparison rules.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `comparisonType` is not Ordinal or OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String, StringComparison) method always returns 0 (zero) to indicate that the match is found at the beginning of the current instance.

In the following example, the IndexOf(String, StringComparison) method is used to find three substrings (a soft hyphen (U+00AD), a soft hyphen followed by "n", and a soft hyphen followed by "m") in two strings. Only one of the strings contains a soft hyphen. If the example is run on the .NET Framework 4 or later, because the soft hyphen is an ignorable character, a culture-sensitive search returns the same value that it would return if the soft hyphen were not included in the search string. An ordinal search, however, successfully finds the soft hyphen in one string and reports that it is absent from the second string.

C#

```
using System;
```

```
public class Example
{
    public static void Main()
    {
        string s1 = "ani\u00ADmal";
        string s2 = "animal";

        Console.WriteLine("Culture-sensitive comparison:");
        // Use culture-sensitive comparison to find the soft hyphen.
        Console.WriteLine(s1.IndexOf("\u00AD",
StringComparison.CurrentCulture));
        Console.WriteLine(s2.IndexOf("\u00AD",
StringComparison.CurrentCulture));

        // Use culture-sensitive comparison to find the soft hyphen
followed by "n".
        Console.WriteLine(s1.IndexOf("\u00ADn",
StringComparison.CurrentCulture));
        Console.WriteLine(s2.IndexOf("\u00ADn",
StringComparison.CurrentCulture));

        // Use culture-sensitive comparison to find the soft hyphen
followed by "m".
        Console.WriteLine(s1.IndexOf("\u00ADm",
StringComparison.CurrentCulture));
        Console.WriteLine(s2.IndexOf("\u00ADm",
StringComparison.CurrentCulture));

        Console.WriteLine("Ordinal comparison:");
        // Use ordinal comparison to find the soft hyphen.
        Console.WriteLine(s1.IndexOf("\u00AD",
StringComparison.Ordinal));
        Console.WriteLine(s2.IndexOf("\u00AD",
StringComparison.Ordinal));

        // Use ordinal comparison to find the soft hyphen followed by
"n".
        Console.WriteLine(s1.IndexOf("\u00ADn",
StringComparison.Ordinal));
        Console.WriteLine(s2.IndexOf("\u00ADn",
StringComparison.Ordinal));

        // Use ordinal comparison to find the soft hyphen followed by
"m".
        Console.WriteLine(s1.IndexOf("\u00ADm",
StringComparison.Ordinal));
        Console.WriteLine(s2.IndexOf("\u00ADm",
StringComparison.Ordinal));

        // The example displays the following output:
        //       Culture-sensitive comparison:
        //       0
        //       0
        //       1
        //       1
```

```
        //      4
        //      3
        //      Ordinal comparison:
        //      3
        //      -1
        //      -1
        //      -1
        //      3
        //      -1
    }
}
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(Char, Int32, Int32)

Reports the zero-based index of the first occurrence of the specified character in this instance. The search starts at a specified character position and examines a specified number of character positions.

C#

```
public int IndexOf (char value, int startIndex, int count);
```

Parameters

`value` Char

A Unicode character to seek.

`startIndex` Int32

The search starting position.

`count` Int32

The number of character positions to examine.

Returns

Int32

The zero-based index position of `value` from the start of the string if that character is found, or -1 if it is not.

Exceptions

ArgumentOutOfRangeException

`count` or `startIndex` is negative.

-or-

`startIndex` is greater than the length of this string.

-or-

`count` is greater than the length of this string minus `startIndex`.

Examples

The following example demonstrates the IndexOf method.

C#

```
// Example for the String.IndexOf( char, int, int ) method.
using System;

class IndexOfCII
{
    public static void Main()
    {
        string br1 =
            "0----+----1----+----2----+----3----+----" +
            "4----+----5----+----6----+----7";
        string br2 =
            "0123456789012345678901234567890123456789" +
            "0123456789012345678901234567890";
        string str =
            "ABCDEFGHI abcdefghi ABCDEFGHI abcdefghi " +
            "ABCDEFGHI abcdefghi ABCDEFGHI";
```

```
        Console.WriteLine(
            "This example of String.IndexOf( char, int, int )\n" +
            "generates the following output." );
        Console.WriteLine(
            "{0}{1}{0}{2}{0}{3}{0}",
            Environment.NewLine, br1, br2, str );

        FindAllChar( 'A', str );
        FindAllChar( 'a', str );
        FindAllChar( 'I', str );
        FindAllChar( 'i', str );
        FindAllChar( '@', str );
        FindAllChar( ' ', str );
    }

    static void FindAllChar( Char target, String searched )
    {
        Console.Write(
            "The character '{0}' occurs at position(s): ",
            target );

        int     startIndex = -1;
        int     hitCount = 0;

        // Search for all occurrences of the target.
        while( true )
        {
            startIndex = searched.IndexOf(
                target, startIndex + 1,
                searched.Length - startIndex - 1 );

            // Exit the loop if the target is not found.
            if( startIndex < 0 )
                break;

            Console.Write( "{0}, ", startIndex );
            hitCount++;
        }

        Console.WriteLine( "occurrences: {0}", hitCount );
    }
}

/*
This example of String.IndexOf( char, int, int )
generates the following output.

0----+----1----+----2----+----3----+----4----+----5----+----6----+----7
01234567890123456789012345678901234567890123456789012345678901234567890
ABCDEFGHI abcdefghi ABCDEFGHI abcdefghi ABCDEFGHI abcdefghi ABCDEFGHI

The character 'A' occurs at position(s): 0, 20, 40, 60, occurrences: 4
The character 'a' occurs at position(s): 10, 30, 50, occurrences: 3
The character 'I' occurs at position(s): 8, 28, 48, 68, occurrences: 4
```

```
The character 'i' occurs at position(s): 18, 38, 58, occurrences: 3
The character '@' occurs at position(s): occurrences: 0
The character ' ' occurs at position(s): 9, 19, 29, 39, 49, 59,
occurrences: 6
*/
```

Remarks

The search begins at `startIndex` and continues to `startIndex` + `count` -1. The character at `startIndex` + `count` is not included in the search.

Index numbering starts from 0 (zero). The `startIndex` parameter can range from 0 to the length of the string instance.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

IndexOf(Char, StringComparison)

Reports the zero-based index of the first occurrence of the specified Unicode character in this string. A parameter specifies the type of search to use for the specified character.

C#

```
public int IndexOf (char value, StringComparison comparisonType);
```

Parameters

`value` Char

The character to seek.

`comparisonType` StringComparison

An enumeration value that specifies the rules for the search.

Returns

Int32

The zero-based index of `value` if that character is found, or -1 if it is not.

Exceptions

ArgumentException

`comparisonType` is not a valid StringComparison value.

Remarks

Index numbering starts from zero.

The `comparisonType` parameter is a StringComparison enumeration member that specifies whether the search for the `value` argument uses the current or invariant culture, is case-sensitive or case-insensitive, or uses word or ordinal comparison rules.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

IndexOf(Char, Int32)

Reports the zero-based index of the first occurrence of the specified Unicode character in this string. The search starts at a specified character position.

```
C#

public int IndexOf (char value, int startIndex);
```

Parameters

`value` Char

A Unicode character to seek.

`startIndex` Int32

The search starting position.

Returns

Int32

The zero-based index position of `value` from the start of the string if that character is found, or -1 if it is not.

Exceptions

ArgumentOutOfRangeException

`startIndex` is less than 0 (zero) or greater than the length of the string.

Examples

The following example demonstrates the IndexOf method.

```C#
string br1 = "0----+----1----+----2----+----3----+----4----+----5----+---
-6----+---";
string br2 =
"01234567890123456789012345678901234567890123456789012345678901234567890123456789012345678";
string str = "Now is the time for all good men to come to the aid of
their country.";
int start;
int at;

Console.WriteLine();
Console.WriteLine("All occurrences of 't' from position 0 to {0}.",
str.Length-1);
Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1, br2,
str);
Console.Write("The letter 't' occurs at position(s): ");

at = 0;
start = 0;
while((start < str.Length) && (at > -1))
{
    at = str.IndexOf('t', start);
    if (at == -1) break;
    Console.Write("{0} ", at);
    start = at+1;
}
Console.WriteLine();

/*
This example produces the following results:

All occurrences of 't' from position 0 to 68.
0----+----1----+----2----+----3----+----4----+----5----+----6----+---
01234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
Now is the time for all good men to come to the aid of their country.

The letter 't' occurs at position(s): 7 11 33 41 44 55 65

*/
```

Remarks

Index numbering starts from 0. The `startIndex` parameter can range from 0 to the length of the string instance. If `startIndex` equals the length of the string instance, the method returns -1.

The search ranges from `startIndex` to the end of the string.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Int32
- Char
- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(String)

Reports the zero-based index of the first occurrence of the specified string in this instance.

C#

```
public int IndexOf (string value);
```

Parameters

`value` String

The string to seek.

Returns

Int32

The zero-based index position of `value` if that string is found, or -1 if it is not. If `value` is Empty, the return value is 0.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example searches for the "n" in "animal". Because string indexes begin at zero rather than one, the IndexOf(String) method indicates that the "n" is at position 1.

C#

```
String str = "animal";
String toFind = "n";
int index = str.IndexOf("n");
Console.WriteLine("Found '{0}' in '{1}' at position {2}",
                toFind, str, index);

// The example displays the following output:
//        Found 'n' in 'animal' at position 1
```

The following example uses the IndexOf method to determine the starting position of an animal name in a sentence. It then uses this position to insert an adjective that describes the animal into the sentence.

C#

```
using System;

public class Example {
    public static void Main()
    {
        string animal1 = "fox";
        string animal2 = "dog";
```

```
        string strTarget = String.Format("The {0} jumps over the {1}.",
                                         animal1, animal2);

        Console.WriteLine("The original string is:{0}{1}{0}",
                          Environment.NewLine, strTarget);

        Console.Write("Enter an adjective (or group of adjectives) " +
                      "to describe the {0}: ==> ", animal1);
        string adj1 = Console.ReadLine();

        Console.Write("Enter an adjective (or group of adjectives) " +
                      "to describe the {0}: ==> ", animal2);
        string adj2 = Console.ReadLine();

        adj1 = adj1.Trim() + " ";
        adj2 = adj2.Trim() + " ";

        strTarget = strTarget.Insert(strTarget.IndexOf(animal1), adj1);
        strTarget = strTarget.Insert(strTarget.IndexOf(animal2), adj2);

        Console.WriteLine("{0}The final string is:{0}{1}",
                          Environment.NewLine, strTarget);
    }
}
// Output from the example might appear as follows:
//       The original string is:
//       The fox jumps over the dog.
//
//       Enter an adjective (or group of adjectives) to describe the fox:
==> bold
//       Enter an adjective (or group of adjectives) to describe the dog:
==> lazy
//
//       The final string is:
//       The bold fox jumps over the lazy dog.
```

Remarks

Index numbering starts from zero.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture. The search begins at the first character position of this instance and continues until the last character position.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String) method always returns 0 (zero) to

indicate that the match is found at the beginning of the current instance. In the following example, the IndexOf(String) method is used to find three substrings (a soft hyphen (U+00AD), a soft hyphen followed by "n", and a soft hyphen followed by "m") in two strings. Only one of the strings contains a soft hyphen. If the example is run on the .NET Framework 4 or later, in each case, because the soft hyphen is an ignorable character, the result is the same as if the soft hyphen had not been included in `value`. When searching for a soft hyphen only, the method returns 0 (zero) to indicate that it has found a match at the beginning of the string.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string s1 = "ani\u00ADmal";
        string s2 = "animal";

        // Find the index of the soft hyphen.
        Console.WriteLine(s1.IndexOf("\u00AD"));
        Console.WriteLine(s2.IndexOf("\u00AD"));

        // Find the index of the soft hyphen followed by "n".
        Console.WriteLine(s1.IndexOf("\u00ADn"));
        Console.WriteLine(s2.IndexOf("\u00ADn"));

        // Find the index of the soft hyphen followed by "m".
        Console.WriteLine(s1.IndexOf("\u00ADm"));
        Console.WriteLine(s2.IndexOf("\u00ADm"));

        // The example displays the following output
        // if run under the .NET Framework 4 or later:
        //       0
        //       0
        //       1
        //       1
        //       4
        //       3
    }
}
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To find the first index of a

substring within a string instance by using the comparison rules of the current culture, signal your intention explicitly by calling the IndexOf(String, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32
- CultureInfo
- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(Char)

Reports the zero-based index of the first occurrence of the specified Unicode character in this string.

C#

```
public int IndexOf (char value);
```

Parameters

`value` Char

A Unicode character to seek.

Returns

Int32

The zero-based index position of `value` if that character is found, or -1 if it is not.

Examples

The following example demonstrates how you can search a String for a character using the IndexOf method.

C#

```
// Create a Unicode string with 5 Greek Alpha characters.
String szGreekAlpha = new String('\u0391',5);

// Create a Unicode string with 3 Greek Omega characters.
String szGreekOmega = "\u03A9\u03A9\u03A9";

String szGreekLetters = String.Concat(szGreekOmega, szGreekAlpha,
                                      szGreekOmega.Clone());

// Display the entire string.
Console.WriteLine("The string: {0}", szGreekLetters);

// The first index of Alpha.
int ialpha = szGreekLetters.IndexOf('\u0391');
// The first index of Omega.
int iomega = szGreekLetters.IndexOf('\u03A9');

Console.WriteLine("First occurrence of the Greek letter Alpha: Index
{0}",
                ialpha);
Console.WriteLine("First occurrence of the Greek letter Omega: Index
{0}",
                iomega);

// The example displays the following output:
//    The string: ΩΩΩΑΑΑΑΑΩΩΩ
//    First occurrence of the Greek letter Alpha: Index 3
//    First occurrence of the Greek letter Omega: Index 0
```

Remarks

Index numbering starts from zero.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOf(String, Int32)

Reports the zero-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position.

C#

```
public int IndexOf (string value, int startIndex);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position.

Returns

Int32

The zero-based index position of `value` from the start of the current instance if that string is found, or -1 if it is not. If `value` is Empty, the return value is `startIndex`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is less than 0 (zero) or greater than the length of this string.

Examples

The following example searches for all occurrences of a specified string within a target string.

C#

```
using System;

public class IndexOfTest {
    public static void Main() {

        string strSource = "This is the string which we will perform the search on";

        Console.WriteLine("The search string is:{0}\"{1}\"{0}", Environment.NewLine, strSource);

        string strTarget = "";
        int found = 0;
        int totFinds = 0;

        do {
            Console.Write("Please enter a search value to look for in the above string (hit Enter to exit) ==> ");
```

```
            strTarget = Console.ReadLine();

            if (strTarget != "") {

                for (int i = 0; i < strSource.Length; i++) {

                    found = strSource.IndexOf(strTarget, i);

                    if (found >= 0) {
                        totFinds++;
                        i = found;
                    }
                    else
                    {
                        break;
                    }
                }
            }
            else
            {
                return;
            }

            Console.WriteLine("{0}The search parameter '{1}' was found
{2} times.{0}",
                    Environment.NewLine, strTarget, totFinds);

            totFinds = 0;
        } while ( true );
    }
}
```

Remarks

Index numbering starts from 0. The `startIndex` parameter can range from 0 to the length of the string instance. If `startIndex` equals the length of the string instance, the method returns -1.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture. The search begins at the `startIndex` character position of this instance and continues until the last character position.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed. If `value` consists only of one or more ignorable characters, the IndexOf(String, Int32) method always returns

`startIndex`, which is the character position at which the search begins. In the following example, the IndexOf(String, Int32) method is used to find the position of a soft hyphen (U+00AD) followed by an "m" in two strings. Only one of the strings contains the required substring. If the example is run on the .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method fails to return the index of the soft hyphen but instead returns the index of the "m".

C#

```
using System;

public class Example
{
    public static void Main()
    {
        string searchString = "\u00ADm";
        string s1 = "ani\u00ADmal" ;
        string s2 = "animal";

        Console.WriteLine(s1.IndexOf(searchString, 2));
        Console.WriteLine(s2.IndexOf(searchString, 2));

        // The example displays the following output:
        //       4
        //       3
    }
}
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To find the first index of a substring that occurs after a particular character position by using the comparison rules of the current culture, signal your intention explicitly by calling the IndexOf(String, Int32, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32
- CultureInfo

- IndexOfAny(Char[])
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IndexOfAny Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters. The method returns -1 if the characters in the array are not found in this instance.

Overloads

Expand table

IndexOfAny(Char[])	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters.
IndexOfAny(Char[], Int32)	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.
IndexOfAny(Char[], Int32, Int32)	Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position and examines a specified number of character positions.

IndexOfAny(Char[])

Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters.

C#

```
public int IndexOfAny (char[] anyOf);
```

Parameters

anyOf Char[]

A Unicode character array containing one or more characters to seek.

Returns

Int32

The zero-based index position of the first occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found.

Exceptions

ArgumentNullException

`anyOf` is `null`.

Examples

The following example finds the first vowel in a string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      char[] chars = { 'a', 'e', 'i', 'o', 'u', 'y',
                       'A', 'E', 'I', 'O', 'U', 'Y' };
      String s = "The long and winding road...";
      Console.WriteLine("The first vowel in \n   {0}\nis found at
position {1}",
                        s, s.IndexOfAny(chars) + 1);
   }
}
// The example displays the following output:
//       The first vowel in
//          The long and winding road...
//       is found at position 3
```

Remarks

Index numbering starts from zero.

The search for `anyOf` is case-sensitive. If `anyOf` is an empty array, the method finds a match at the beginning of the string (that is, at index zero).

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- IndexOf(Char)
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOfAny(Char[], Int32)

Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.

C#

```
public int IndexOfAny (char[] anyOf, int startIndex);
```

Parameters

`anyOf` Char[]

A Unicode character array containing one or more characters to seek.

`startIndex` Int32

The search starting position.

Returns

Int32

The zero-based index position of the first occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found.

Exceptions

ArgumentNullException

`anyOf` is `null`.

ArgumentOutOfRangeException

`startIndex` is negative.

-or-

`startIndex` is greater than the number of characters in this instance.

Examples

The following example finds the index of the occurrence of any character of the string "is" within a substring of another string.

C#

```
// Sample for String.IndexOfAny(Char[], Int32)
using System;

class Sample {
    public static void Main()
    {
    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
```

```
    string target = "is";
    char[] anyOf = target.ToCharArray();

    start = str.Length/2;
    Console.WriteLine();
    Console.WriteLine("The first character occurrence from position {0}
to {1}.",
                            start, str.Length-1);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("A character in '{0}' occurs at position: ", target);

    at = str.IndexOfAny(anyOf, start);
    if (at > -1)
        Console.Write(at);
    else
        Console.Write("(not found)");
    Console.WriteLine();
    }
}
/*

The first character occurrence from position 33 to 66.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

A character in 'is' occurs at position: 49

*/
```

Remarks

Index numbering starts from zero. The `startIndex` parameter can range from 0 to one less than the length of the string instance.

The search ranges from `startIndex` to the end of the string.

The search for `anyOf` is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar value are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- Array
- IndexOf(Char)
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IndexOfAny(Char[], Int32, Int32)

Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position and examines a specified number of character positions.

C#

```
public int IndexOfAny (char[] anyOf, int startIndex, int count);
```

Parameters

anyOf Char[]

A Unicode character array containing one or more characters to seek.

startIndex Int32

The search starting position.

`count` Int32

The number of character positions to examine.

Returns

Int32

The zero-based index position of the first occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found.

Exceptions

ArgumentNullException

`anyOf` is `null`.

ArgumentOutOfRangeException

`count` or `startIndex` is negative.

-or-

`count` + `startIndex` is greater than the number of characters in this instance.

Examples

The following example finds the index of the occurrence of any character of the string "aid" within a substring of another string.

C#

```
// Sample for String.IndexOfAny(Char[], Int32, Int32)
using System;

class Sample {
    public static void Main()
    {
    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    int count;
    string target = "aid";
    char[] anyOf = target.ToCharArray();
```

```
    start = (str.Length-1)/3;
    count = (str.Length-1)/4;
    Console.WriteLine();
    Console.WriteLine("The first character occurrence from position {0}
for {1} characters.", start, count);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("A character in '{0}' occurs at position: ", target);

    at = str.IndexOfAny(anyOf, start, count);
    if (at > -1)
        Console.Write(at);
    else
        Console.Write("(not found)");
    Console.WriteLine();
    }
}
/*

The first character occurrence from position 22 for 16 characters.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

A character in 'aid' occurs at position: 27

*/
```

Remarks

The search begins at `startIndex` and continues to `startIndex` + `count` -1. The character at `startIndex` + `count` is not included in the search.

Index numbering starts from zero. The `startIndex` parameter can range from 0 to one less than the length of the string instance.

The search for `anyOf` is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar value are the same. To perform a culture-sensitive search, use the CompareInfo.IndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- Array
- IndexOf(Char)
- LastIndexOf(Char)
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Insert(Int32, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string in which a specified string is inserted at a specified index position in this instance.

```
C#
public string Insert (int startIndex, string value);
```

Parameters

`startIndex` Int32

The zero-based index position of the insertion.

`value` String

The string to insert.

Returns

String

A new string that is equivalent to this instance, but with `value` inserted at position `startIndex`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` is negative or greater than the length of this instance.

Examples

The following example inserts a space character in the fourth character position (the character at index 3) of a string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String original = "aaabbb";
      Console.WriteLine("The original string: '{0}'", original);
      String modified = original.Insert(3, " ");
      Console.WriteLine("The modified string: '{0}'", modified);
   }
}
// The example displays the following output:
//     The original string: 'aaabbb'
//     The modified string: 'aaa bbb'
```

The following console application prompts the users to enter one or more adjectives to describe two animals. It then calls the Insert method to insert the text entered by the user into a string.

C#

```
using System;

public class Example {
    public static void Main()
    {
        string animal1 = "fox";
        string animal2 = "dog";

        string strTarget = String.Format("The {0} jumps over the {1}.",
                                         animal1, animal2);

        Console.WriteLine("The original string is:{0}{1}{0}",
                          Environment.NewLine, strTarget);

        Console.Write("Enter an adjective (or group of adjectives) " +
                      "to describe the {0}: ==> ", animal1);
        string adj1 = Console.ReadLine();

        Console.Write("Enter an adjective (or group of adjectives) " +
                      "to describe the {0}: ==> ", animal2);
        string adj2 = Console.ReadLine();

        adj1 = adj1.Trim() + " ";
        adj2 = adj2.Trim() + " ";
```

```
        strTarget = strTarget.Insert(strTarget.IndexOf(animal1), adj1);
        strTarget = strTarget.Insert(strTarget.IndexOf(animal2), adj2);

        Console.WriteLine("{0}The final string is:{0}{1}",
                          Environment.NewLine, strTarget);
    }
}
// Output from the example might appear as follows:
//       The original string is:
//       The fox jumps over the dog.
//
//       Enter an adjective (or group of adjectives) to describe the fox:
==> bold
//       Enter an adjective (or group of adjectives) to describe the dog:
==> lazy
//
//       The final string is:
//       The bold fox jumps over the lazy dog.
```

Remarks

If `startIndex` is equal to the length of this instance, `value` is appended to the end of this instance.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which `value` is inserted into the current instance.

For example, the return value of `"abc".Insert(2, "XYZ")` is "abXYZc".

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int32
- Concat(Object)
- CopyTo(Int32, Char[], Int32, Int32)
- Insert(Int32, String)
- Join(String, String[])
- Remove(Int32, Int32)
- Replace(Char, Char)
- Split(Char[])
- Substring(Int32)
- Trim(Char[])

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Intern(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves the system's reference to the specified String.

C#

```
public static string Intern (string str);
```

Parameters

`str` String

A string to search for in the intern pool.

Returns

String

The system's reference to `str`, if it is interned; otherwise, a new reference to a string with the value of `str`.

Exceptions

ArgumentNullException

`str` is `null`.

Examples

The following example uses three strings that are equal in value to determine whether a newly created string and an interned string are equal.

C#

```
// Sample for String.Intern(String)
using System;
using System.Text;
```

```
class Sample
{
    public static void Main()
    {
        string s1 = "MyTest";
        string s2 = new
StringBuilder().Append("My").Append("Test").ToString();
        string s3 = String.Intern(s2);
        Console.WriteLine($"s1 == {s1}");
        Console.WriteLine($"s2 == {s2}");
        Console.WriteLine($"s3 == {s3}");
        Console.WriteLine($"Is s2 the same reference as s1?: {(Object)s2 ==
(Object)s1}");
        Console.WriteLine($"Is s3 the same reference as s1?: {(Object)s3 ==
(Object)s1}");
    }
}
/*
This example produces the following results:
s1 == MyTest
s2 == MyTest
s3 == MyTest
Is s2 the same reference as s1?: False
Is s3 the same reference as s1?: True
*/
```

Remarks

For more information about this API, see Supplemental API remarks for String.Intern.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsInterned(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.IsInterned(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves a reference to a specified String.

C#

```
public static string? IsInterned (string str);
```

Parameters

`str` String

The string to search for in the intern pool.

Returns

String

A reference to `str` if it is in the common language runtime intern pool; otherwise, `null`.

Exceptions

ArgumentNullException

`str` is `null`.

Examples

The following example demonstrates that literal strings are interned automatically by the compiler.

C#

```
// Sample for String.IsInterned(String)
using System;
using System.Text;
using System.Runtime.CompilerServices;
```

```
// In the .NET Framework 2.0 the following attribute declaration allows you
to
// avoid the use of the interning when you use NGEN.exe to compile an
assembly
// to the native image cache.
[assembly: CompilationRelaxations(CompilationRelaxations.NoStringInterning)]
class Sample
{
    public static void Main()
    {
        // String str1 is known at compile time, and is automatically
interned.
        String str1 = "abcd";

        // Constructed string, str2, is not explicitly or automatically
interned.
        String str2 = new
StringBuilder().Append("wx").Append("yz").ToString();
        Console.WriteLine();
        Test(1, str1);
        Test(2, str2);
    }

    public static void Test(int sequence, String str)
    {
        Console.Write("{0}) The string, '", sequence);
        String strInterned = String.IsInterned(str);
        if (strInterned == null)
            Console.WriteLine("{0}', is not interned.", str);
        else
            Console.WriteLine("{0}', is interned.", strInterned);
    }
}

//This example produces the following results:

//1) The string, 'abcd', is interned.
//2) The string, 'wxyz', is not interned.

//If you use NGEN.exe to compile the assembly to the native image cache,
this
//example produces the following results:

//1) The string, 'abcd', is not interned.
//2) The string, 'wxyz', is not interned.
```

Remarks

The common language runtime automatically maintains a table, called the intern pool, which contains a single instance of each unique literal string constant declared in a

program, as well as any unique instance of String you add programmatically by calling the Intern method.

The intern pool conserves string storage. If you assign a literal string constant to several variables, each variable is set to reference the same constant in the intern pool instead of referencing several different instances of String that have identical values.

This method looks up `str` in the intern pool. If `str` has already been interned, a reference to that instance is returned; otherwise, `null` is returned.

Compare this method to the Intern method.

This method does not return a Boolean value. If you call the method because you want a Boolean value that indicates whether a particular string is interned, you can use code such as the following.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string str1 = "a";
      string str2 = str1 + "b";
      string str3 = str2 + "c";
      string[] strings = { "value", "part1" + "_" + "part2", str3,
                           String.Empty, null };
      foreach (var value in strings) {
         if (value == null) continue;

         bool interned = String.IsInterned(value) != null;
         if (interned)
            Console.WriteLine("'{0}' is in the string intern pool.",
                              value);
         else
            Console.WriteLine("'{0}' is not in the string intern pool.",
                              value);
      }
   }
}
// The example displays the following output:
//       'value' is in the string intern pool.
//       'part1_part2' is in the string intern pool.
//       'abc' is not in the string intern pool.
//       '' is in the string intern pool.
```

Note

You can override the use of the intern pool when you use **Ngen.exe (Native Image Generator)** to install an assembly to the native image cache on a local computer. For more information, see Performance Considerations in the Remarks section for the **Intern** property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Intern(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IsNormalized Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether this string is in a particular Unicode normalization form.

Overloads

Expand table

IsNormalized(NormalizationForm)	Indicates whether this string is in the specified Unicode normalization form.
IsNormalized()	Indicates whether this string is in Unicode normalization form C.

Examples

The following example determines whether a string is successfully normalized to various normalization forms.

C#

```
using System;
using System.Text;

class Example
{
    public static void Main()
    {
       // Character c; combining characters acute and cedilla; character 3/4
       string s1 = new String( new char[] {'\u0063', '\u0301', '\u0327',
'\u00BE'});
       string s2 = null;
       string divider = new String('-', 80);
       divider = String.Concat(Environment.NewLine, divider,
Environment.NewLine);

       Show("s1", s1);
       Console.WriteLine();
```

```
        Console.WriteLine("U+0063 = LATIN SMALL LETTER C");
        Console.WriteLine("U+0301 = COMBINING ACUTE ACCENT");
        Console.WriteLine("U+0327 = COMBINING CEDILLA");
        Console.WriteLine("U+00BE = VULGAR FRACTION THREE QUARTERS");
        Console.WriteLine(divider);

        Console.WriteLine("A1) Is s1 normalized to the default form (Form
C)?: {0}",
                                    s1.IsNormalized());
        Console.WriteLine("A2) Is s1 normalized to Form C?:  {0}",

s1.IsNormalized(NormalizationForm.FormC));
        Console.WriteLine("A3) Is s1 normalized to Form D?:  {0}",

s1.IsNormalized(NormalizationForm.FormD));
        Console.WriteLine("A4) Is s1 normalized to Form KC?: {0}",

s1.IsNormalized(NormalizationForm.FormKC));
        Console.WriteLine("A5) Is s1 normalized to Form KD?: {0}",

s1.IsNormalized(NormalizationForm.FormKD));

        Console.WriteLine(divider);

        Console.WriteLine("Set string s2 to each normalized form of string
s1.");
        Console.WriteLine();
        Console.WriteLine("U+1E09 = LATIN SMALL LETTER C WITH CEDILLA AND
ACUTE");
        Console.WriteLine("U+0033 = DIGIT THREE");
        Console.WriteLine("U+2044 = FRACTION SLASH");
        Console.WriteLine("U+0034 = DIGIT FOUR");
        Console.WriteLine(divider);

        s2 = s1.Normalize();
        Console.Write("B1) Is s2 normalized to the default form (Form C)?:
");
        Console.WriteLine(s2.IsNormalized());
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormC);
        Console.Write("B2) Is s2 normalized to Form C?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormC));
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormD);
        Console.Write("B3) Is s2 normalized to Form D?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormD));
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormKC);
        Console.Write("B4) Is s2 normalized to Form KC?: ");
```

```
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormKC));
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormKD);
        Console.Write("B5) Is s2 normalized to Form KD?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormKD));
        Show("s2", s2);
        Console.WriteLine();
    }

    private static void Show(string title, string s)
    {
        Console.Write("Characters in string {0} = ", title);
        foreach(short x in s) {
            Console.Write("{0:X4} ", x);
        }
        Console.WriteLine();
    }
}
/*
This example produces the following results:

Characters in string s1 = 0063 0301 0327 00BE

U+0063 = LATIN SMALL LETTER C
U+0301 = COMBINING ACUTE ACCENT
U+0327 = COMBINING CEDILLA
U+00BE = VULGAR FRACTION THREE QUARTERS

--------------------------------------------------------------------------
----

A1) Is s1 normalized to the default form (Form C)?: False
A2) Is s1 normalized to Form C?:  False
A3) Is s1 normalized to Form D?:  False
A4) Is s1 normalized to Form KC?: False
A5) Is s1 normalized to Form KD?: False

--------------------------------------------------------------------------
----

Set string s2 to each normalized form of string s1.

U+1E09 = LATIN SMALL LETTER C WITH CEDILLA AND ACUTE
U+0033 = DIGIT THREE
U+2044 = FRACTION SLASH
U+0034 = DIGIT FOUR

--------------------------------------------------------------------------
----

B1) Is s2 normalized to the default form (Form C)?: True
Characters in string s2 = 1E09 00BE
```

```
B2) Is s2 normalized to Form C?: True
Characters in string s2 = 1E09 00BE

B3) Is s2 normalized to Form D?: True
Characters in string s2 = 0063 0327 0301 00BE

B4) Is s2 normalized to Form KC?: True
Characters in string s2 = 1E09 0033 2044 0034

B5) Is s2 normalized to Form KD?: True
Characters in string s2 = 0063 0327 0301 0033 2044 0034

*/
```

IsNormalized(NormalizationForm)

Indicates whether this string is in the specified Unicode normalization form.

```
C#

public bool IsNormalized (System.Text.NormalizationForm
normalizationForm);
```

Parameters

`normalizationForm` NormalizationForm

A Unicode normalization form.

Returns

Boolean

`true` if this string is in the normalization form specified by the `normalizationForm` parameter; otherwise, `false`.

Exceptions

ArgumentException

The current instance contains invalid Unicode characters.

Remarks

Some Unicode characters have multiple equivalent binary representations consisting of sets of combining and/or composite Unicode characters. The existence of multiple

representations for a single character complicates searching, sorting, matching, and other operations.

The Unicode standard defines a process called normalization that returns one binary representation when given any of the equivalent binary representations of a character. Normalization can be performed with several algorithms, called normalization forms, that obey different rules. .NET currently supports normalization forms C, D, KC, and KD.

For a description of supported Unicode normalization forms, see System.Text.NormalizationForm.

Notes to Callers

The IsNormalized method returns `false` as soon as it encounters the first non-normalized character in a string. Therefore, if a string contains non-normalized characters followed by invalid Unicode characters, the Normalize method will throw an ArgumentException although IsNormalized returns `false`.

See also

- Normalize()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsNormalized()

Indicates whether this string is in Unicode normalization form C.

C#

```
public bool IsNormalized ();
```

Returns

Boolean

`true` if this string is in normalization form C; otherwise, `false`.

Exceptions

ArgumentException

The current instance contains invalid Unicode characters.

Remarks

Some Unicode characters have multiple equivalent binary representations consisting of sets of combining and/or composite Unicode characters. The existence of multiple representations for a single character complicates searching, sorting, matching, and other operations.

The Unicode standard defines a process called normalization that returns one binary representation when given any of the equivalent binary representations of a character. Normalization can be performed with several algorithms, called normalization forms, that obey different rules. .NET currently supports normalization forms C, D, KC, and KD.

For a description of supported Unicode normalization forms, see System.Text.NormalizationForm.

Notes to Callers

The IsNormalized method returns `false` as soon as it encounters the first non-normalized character in a string. Therefore, if a string contains non-normalized characters followed by invalid Unicode characters, the Normalize method will throw an ArgumentException although IsNormalized returns `false`.

See also

- Normalize()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IsNullOrEmpty(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the specified string is `null` or an empty string ("").

C#

```
public static bool IsNullOrEmpty (string? value);
```

Parameters

`value` String

The string to test.

Returns

Boolean

`true` if the `value` parameter is `null` or an empty string (""); otherwise, `false`.

Examples

The following example examines three strings and determines whether each string has a value, is an empty string, or is `null`.

C#

```
string s1 = "abcd";
string s2 = "";
string s3 = null;

Console.WriteLine("String s1 {0}.", Test(s1));
Console.WriteLine("String s2 {0}.", Test(s2));
Console.WriteLine("String s3 {0}.", Test(s3));

String Test(string s)
{
if (String.IsNullOrEmpty(s))
```

```
    return "is null or empty";
else
    return String.Format("(\"{0}\") is neither null nor empty", s);
}

// The example displays the following output:
//       String s1 ("abcd") is neither null nor empty.
//       String s2 is null or empty.
//       String s3 is null or empty.
```

Remarks

For more information about this API, see Supplemental API remarks for String.IsNullOrEmpty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNullOrWhiteSpace(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IsNullOrWhiteSpace(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified string is `null`, empty, or consists only of white-space characters.

```
C#

public static bool IsNullOrWhiteSpace (string? value);
```

Parameters

`value` String

The string to test.

Returns

Boolean

`true` if the `value` parameter is `null` or Empty, or if `value` consists exclusively of white-space characters.

Examples

The following example creates a string array, and then passes each element of the array to the IsNullOrWhiteSpace method.

```
C#

using System;

public class Example
{
   public static void Main()
   {
```

```
        string[] values = { null, String.Empty, "ABCDE",
                            new String(' ', 20), "  \t   ",
                            new String('\u2000', 10) };
        foreach (string value in values)
           Console.WriteLine(String.IsNullOrWhiteSpace(value));
    }
}
// The example displays the following output:
//       True
//       True
//       False
//       True
//       True
//       True
```

Remarks

IsNullOrWhiteSpace is a convenience method that is similar to the following code, except that it offers superior performance:

```
C#

return String.IsNullOrEmpty(value) || value.Trim().Length == 0;
```

White-space characters are defined by the Unicode standard. The IsNullOrWhiteSpace method interprets any character that returns a value of `true` when it is passed to the Char.IsWhiteSpace method as a white-space character.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNullOrEmpty(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.Join Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Concatenates the elements of a specified array or the members of a collection, using the specified separator between each element or member.

Overloads

Expand table

Join(Char, Object[])	Concatenates the string representations of an array of objects, using the specified separator between each member.
Join(Char, String[])	Concatenates an array of strings, using the specified separator between each member.
Join(String, IEnumerable<String>)	Concatenates the members of a constructed IEnumerable<T> collection of type String, using the specified separator between each member.
Join(String, Object[])	Concatenates the elements of an object array, using the specified separator between each element.
Join(String, String[])	Concatenates all the elements of a string array, using the specified separator between each element.
Join(Char, String[], Int32, Int32)	Concatenates an array of strings, using the specified separator between each member, starting with the element in `value` located at the `startIndex` position, and concatenating up to `count` elements.
Join(String, String[], Int32, Int32)	Concatenates the specified elements of a string array, using the specified separator between each element.
Join<T>(Char, IEnumerable<T>)	Concatenates the members of a collection, using the specified separator between each member.
Join<T>(String, IEnumerable<T>)	Concatenates the members of a collection, using the specified separator between each member.

Join(Char, Object[])

Concatenates the string representations of an array of objects, using the specified separator between each member.

```
C#
public static string Join (char separator, params object?[] values);
```

Parameters

`separator` Char

The character to use as a separator. `separator` is included in the returned string only if `value` has more than one element.

`values` Object[]

An array of objects whose string representations will be concatenated.

Returns

String

A string that consists of the elements of `values` delimited by the `separator` character.

-or-

Empty if `values` has zero elements.

Exceptions

ArgumentNullException

`value` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Join(Char, String[])

Concatenates an array of strings, using the specified separator between each member.

```
C#
public static string Join (char separator, params string?[] value);
```

Parameters

`separator` Char

The character to use as a separator. `separator` is included in the returned string only if `value` has more than one element.

`value` String[]

An array of strings to concatenate.

Returns

String

A string that consists of the elements of `value` delimited by the `separator` character.

-or-

Empty if `value` has zero elements.

Exceptions

ArgumentNullException

`value` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Join(String, IEnumerable<String>)

Concatenates the members of a constructed IEnumerable<T> collection of type String, using the specified separator between each member.

C#

```
public static string Join (string? separator,
System.Collections.Generic.IEnumerable<string?> values);
```

Parameters

`separator` String

The string to use as a separator. `separator` is included in the returned string only if `values` has more than one element.

`values` IEnumerable<String>

A collection that contains the strings to concatenate.

Returns

String

A string that consists of the elements of `values` delimited by the `separator` string.

-or-

Empty if `values` has zero elements.

Exceptions

ArgumentNullException

`values` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Examples

The following example uses the Sieve of Eratosthenes algorithm to calculate the prime numbers that are less than or equal to 100. It assigns the result to a List<T> object of type String, which it then passes to the Join(String, IEnumerable<String>) method.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      int maxPrime = 100;
      List<int> primes = GetPrimes(maxPrime);
      Console.WriteLine("Primes less than {0}:", maxPrime);
      Console.WriteLine("   {0}", String.Join(" ", primes));
   }

   private static List<int> GetPrimes(int maxPrime)
   {
      Array values = Array.CreateInstance(typeof(int),
                              new int[] { maxPrime - 1}, new int[] { 2
});
      // Use Sieve of Eratosthenes to determine prime numbers.
      for (int ctr = values.GetLowerBound(0); ctr <= (int)
Math.Ceiling(Math.Sqrt(values.GetUpperBound(0))); ctr++)
      {

         if ((int) values.GetValue(ctr) == 1) continue;

         for (int multiplier = ctr; multiplier <=  maxPrime / 2;
multiplier++)
            if (ctr * multiplier <= maxPrime)
               values.SetValue(1, ctr * multiplier);
      }

      List<int> primes = new List<int>();
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++)
         if ((int) values.GetValue(ctr) == 0)
```

```
            primes.Add(ctr);
        return primes;
    }
}
// The example displays the following output:
//    Primes less than 100:
//       2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
```

Remarks

If `separator` is `null`, an empty string (String.Empty) is used instead. If any member of `values` is `null`, an empty string is used instead.

Join(String, IEnumerable<String>) is a convenience method that lets you concatenate each element in an `IEnumerable(Of String)` collection without first converting the elements to a string array. It is particularly useful with Language-Integrated Query (LINQ) query expressions. The following example passes a `List(Of String)` object that contains either the uppercase or lowercase letters of the alphabet to a lambda expression that selects letters that are equal to or greater than a particular letter (which, in the example, is "M"). The `IEnumerable(Of String)` collection returned by the Enumerable.Where method is passed to the Join(String, IEnumerable<String>) method to display the result as a single string.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Example
{
   public static void Main()
   {
      string output = String.Join(" ", GetAlphabet(true).Where( letter =>
                      letter.CompareTo("M") >= 0));
      Console.WriteLine(output);
   }

   private static List<string> GetAlphabet(bool upper)
   {
      List<string> alphabet = new List<string>();
      int charValue = upper ? 65 : 97;
      for (int ctr = 0; ctr <= 25; ctr++)
         alphabet.Add(((char)(charValue + ctr)).ToString());
      return alphabet;
   }
}
```

```
// The example displays the following output:
//      M N O P Q R S T U V W X Y Z
```

See also

- Split(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Join(String, Object[])

Concatenates the elements of an object array, using the specified separator between each element.

C#

```
public static string Join (string? separator, params object?[] values);
```

Parameters

`separator` String

The string to use as a separator. `separator` is included in the returned string only if `values` has more than one element.

`values` Object[]

An array that contains the elements to concatenate.

Returns

String

A string that consists of the elements of `values` delimited by the `separator` string.

-or-

Empty if `values` has zero elements.

-or-

.NET Framework only: Empty if the first element of `values` is `null`.

Exceptions

ArgumentNullException

`values` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Examples

The following example uses the Sieve of Eratosthenes algorithm to calculate the prime numbers that are less than or equal to 100. It assigns the result to a integer array, which it then passes to the Join(String, Object[]) method.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      int maxPrime = 100;
      int[] primes = GetPrimes(maxPrime);
      Console.WriteLine("Primes less than {0}:", maxPrime);
      Console.WriteLine("   {0}", String.Join(" ", primes));
   }

   private static int[] GetPrimes(int maxPrime)
   {
      Array values = Array.CreateInstance(typeof(int),
                                          new int[] { maxPrime - 1}, new int[] { 2
```

```
});
      // Use Sieve of Eratosthenes to determine prime numbers.
      for (int ctr = values.GetLowerBound(0); ctr <= (int)
Math.Ceiling(Math.Sqrt(values.GetUpperBound(0))); ctr++)
      {

         if ((int) values.GetValue(ctr) == 1) continue;

         for (int multiplier = ctr; multiplier <=  maxPrime / 2;
multiplier++)
            if (ctr * multiplier <= maxPrime)
               values.SetValue(1, ctr * multiplier);
      }

      List<int> primes = new List<int>();
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++)
         if ((int) values.GetValue(ctr) == 0)
            primes.Add(ctr);
      return primes.ToArray();
   }
}
// The example displays the following output:
//    Primes less than 100:
//       2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
```

Remarks

If `separator` is `null` or if any element of `values` other than the first element is `null`, an empty string (String.Empty) is used instead. See the Notes for Callers section if the first element of `values` is `null`.

Join(String, Object[]) is a convenience method that lets you concatenate each element in an object array without explicitly converting its elements to strings. The string representation of each object in the array is derived by calling that object's `ToString` method.

Notes to Callers

.NET Framework only: If the first element of `values` is `null`, the Join(String, Object[]) method does not concatenate the elements in `values` but instead returns Empty. A number of workarounds for this issue are available. The easiest is to assign a value of Empty to the first element of the array, as the following example shows.

```
C#
```

```
object[] values = { null, "Cobb", 4189, 11434, .366 };
if (values[0] == null) values[0] = String.Empty;
Console.WriteLine(String.Join("|", values));

// The example displays the following output:
//       |Cobb|4189|11434|0.366
```

See also

- Split(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Join(String, String[])

Concatenates all the elements of a string array, using the specified separator between each element.

C#

```
public static string Join (string? separator, params string?[] value);
```

Parameters

`separator` String

The string to use as a separator. `separator` is included in the returned string only if `value` has more than one element.

`value` String[]

An array that contains the elements to concatenate.

Returns

String

A string that consists of the elements in `value` delimited by the `separator` string.

-or-

Empty if `values` has zero elements.

Exceptions

ArgumentNullException

`value` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Examples

The following example demonstrates the Join method.

C#

```
using System;

public class JoinTest
{
    public static void Main()
    {
        Console.WriteLine(MakeLine(0, 5, ", "));
        Console.WriteLine(MakeLine(1, 6, "  "));
        Console.WriteLine(MakeLine(9, 9, ": "));
        Console.WriteLine(MakeLine(4, 7, "< "));
    }

    private static string MakeLine(int initVal, int multVal, string sep)
    {
        string [] sArr = new string [10];

        for (int i = initVal; i < initVal + 10; i++)
            sArr[i - initVal] = String.Format("{0,-3}", i * multVal);
```

```
        return String.Join(sep, sArr);
    }
}
// The example displays the following output:
//       0  , 5  , 10 , 15 , 20 , 25 , 30 , 35 , 40 , 45
//       6    12   18   24   30   36   42   48   54   60
//       81 : 90 : 99 : 108: 117: 126: 135: 144: 153: 162
//       28 < 35 < 42 < 49 < 56 < 63 < 70 < 77 < 84 < 91
```

Remarks

For example, if `separator` is ", " and the elements of `value` are "apple", "orange", "grape", and "pear", `Join(separator, value)` returns "apple, orange, grape, pear".

If `separator` is `null`, an empty string (String.Empty) is used instead. If any element in `value` is `null`, an empty string is used instead.

See also

- Concat(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Join(Char, String[], Int32, Int32)

Concatenates an array of strings, using the specified separator between each member, starting with the element in `value` located at the `startIndex` position, and concatenating up to `count` elements.

```
C#

public static string Join (char separator, string?[] value, int
startIndex, int count);
```

Parameters

separator Char

Concatenates an array of strings, using the specified separator between each member, starting with the element located at the specified index and including a specified number of elements.

value String[]

An array of strings to concatenate.

startIndex Int32

The first item in `value` to concatenate.

count Int32

The number of elements from `value` to concatenate, starting with the element in the `startIndex` position.

Returns

String

A string that consists of `count` elements of `value` starting at `startIndex` delimited by the `separator` character.

-or-

Empty if `count` is zero.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `count` are negative.

-or-

`startIndex` is greater than the length of `value` - `count`.

OutOfMemoryException
The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Join(String, String[], Int32, Int32)

Concatenates the specified elements of a string array, using the specified separator between each element.

C#

```
public static string Join (string? separator, string?[] value, int
startIndex, int count);
```

Parameters

`separator` String

The string to use as a separator. `separator` is included in the returned string only if `value` has more than one element.

`value` String[]

An array that contains the elements to concatenate.

`startIndex` Int32

The first element in `value` to use.

`count` Int32

The number of elements of `value` to use.

Returns

String

A string that consists of `count` elements of `value` starting at `startIndex` delimited by the `separator` character.

-or-

Empty if `count` is zero.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`startIndex` or `count` is less than 0.

-or-

`startIndex` plus `count` is greater than the number of elements in `value`.

OutOfMemoryException

Out of memory.

Examples

The following example concatenates two elements from an array of names of fruit.

C#

```
String[] val = {"apple", "orange", "grape", "pear"};
String sep   = ", ";
String result;

Console.WriteLine("sep = '{0}'", sep);
Console.WriteLine("val[] = {{'{0}' '{1}' '{2}' '{3}'}}", val[0], val[1],
val[2], val[3]);
result = String.Join(sep, val, 1, 2);
Console.WriteLine("String.Join(sep, val, 1, 2) = '{0}'", result);

// This example produces the following results:
// sep = ', '
// val[] = {'apple' 'orange' 'grape' 'pear'}
// String.Join(sep, val, 1, 2) = 'orange, grape'
```

Remarks

For example, if `separator` is ", " and the elements of `value` are "apple", "orange", "grape", and "pear", `Join(separator, value, 1, 2)` returns "orange, grape".

If `separator` is `null`, an empty string (String.Empty) is used instead. If any element in `value` is `null`, an empty string is used instead.

See also

- Split(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Join<T>(Char, IEnumerable<T>)

Concatenates the members of a collection, using the specified separator between each member.

C#

```
public static string Join<T> (char separator,
System.Collections.Generic.IEnumerable<T> values);
```

Type Parameters

T

The type of the members of `values`.

Parameters

`separator` Char

The character to use as a separator. `separator` is included in the returned string only if `values` has more than one element.

`values` IEnumerable<T>

A collection that contains the objects to concatenate.

Returns

String

A string that consists of the members of `values` delimited by the `separator` character.

-or-

Empty if `values` has no elements.

Exceptions

ArgumentNullException

`values` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Join<T>(String, IEnumerable<T>)

Concatenates the members of a collection, using the specified separator between each member.

```
C#

public static string Join<T> (string? separator,
System.Collections.Generic.IEnumerable<T> values);
```

Type Parameters

T

The type of the members of `values`.

Parameters

separator String

The string to use as a separator. `separator` is included in the returned string only if `values` has more than one element.

values IEnumerable<T>

A collection that contains the objects to concatenate.

Returns

String

A string that consists of the elements of `values` delimited by the `separator` string.

-or-

Empty if `values` has no elements.

Exceptions

ArgumentNullException

`values` is `null`.

OutOfMemoryException

The length of the resulting string overflows the maximum allowed length (Int32.MaxValue).

Examples

The following example uses the Sieve of Eratosthenes algorithm to calculate the prime numbers that are less than or equal to 100. It assigns the result to a List<T> object of type integer, which it then passes to the Join<T>(String, IEnumerable<T>) method.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      int maxPrime = 100;
      List<int> primes = GetPrimes(maxPrime);
      Console.WriteLine("Primes less than {0}:", maxPrime);
      Console.WriteLine("   {0}", String.Join(" ", primes));
   }

   private static List<int> GetPrimes(int maxPrime)
   {
      Array values = Array.CreateInstance(typeof(int),
                              new int[] { maxPrime - 1}, new int[] { 2
});
      // Use Sieve of Eratosthenes to determine prime numbers.
      for (int ctr = values.GetLowerBound(0); ctr <= (int)
Math.Ceiling(Math.Sqrt(values.GetUpperBound(0))); ctr++)
      {

         if ((int) values.GetValue(ctr) == 1) continue;

         for (int multiplier = ctr; multiplier <=  maxPrime / 2;
multiplier++)
            if (ctr * multiplier <= maxPrime)
               values.SetValue(1, ctr * multiplier);
      }

      List<int> primes = new List<int>();
      for (int ctr = values.GetLowerBound(0); ctr <=
values.GetUpperBound(0); ctr++)
         if ((int) values.GetValue(ctr) == 0)
            primes.Add(ctr);
      return primes;
   }
}
// The example displays the following output:
//    Primes less than 100:
//       2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
```

Remarks

If `separator` is `null`, an empty string (String.Empty) is used instead. If any member of `values` is `null`, an empty string is used instead.

Join<T>(String, IEnumerable<T>) is a convenience method that lets you concatenate each member of an IEnumerable<T> collection without first converting them to strings. The string representation of each object in the IEnumerable<T> collection is derived by calling that object's `ToString` method.

This method is particular useful with Language-Integrated Query (LINQ) query expressions. For example, the following code defines a very simple `Animal` class that contains the name of an animal and the order to which it belongs. It then defines a List<T> object that contains a number of `Animal` objects. The Enumerable.Where extension method is called to extract the `Animal` objects whose `Order` property equals "Rodent". The result is passed to the Join<T>(String, IEnumerable<T>) method.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Animal
{
   public string Kind;
   public string Order;

   public Animal(string kind, string order)
   {
      this.Kind = kind;
      this.Order = order;
   }

   public override string ToString()
   {
      return this.Kind;
   }
}

public class Example
{
   public static void Main()
   {
      List<Animal> animals = new List<Animal>();
      animals.Add(new Animal("Squirrel", "Rodent"));
      animals.Add(new Animal("Gray Wolf", "Carnivora"));
      animals.Add(new Animal("Capybara", "Rodent"));
```

```
        string output = String.Join(" ", animals.Where( animal =>
                                (animal.Order == "Rodent")));
        Console.WriteLine(output);
    }
}
// The example displays the following output:
//      Squirrel Capybara
```

See also

- Split(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.LastIndexOf Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports the zero-based index position of the last occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.

Overloads

Expand table

LastIndexOf(String, Int32, Int32, StringComparison)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for the specified number of character positions. A parameter specifies the type of comparison to perform when searching for the specified string.
LastIndexOf(String, Int32, StringComparison)	Reports the zero-based index of the last occurrence of a specified string within the current String object. The search starts at a specified character position and proceeds backward toward the beginning of the string. A parameter specifies the type of comparison to perform when searching for the specified string.
LastIndexOf(String, Int32, Int32)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.
LastIndexOf(Char, Int32, Int32)	Reports the zero-based index position of the last occurrence of the specified Unicode character in a substring within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.
LastIndexOf(String, Int32)	Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.

LastIndexOf(Char, Int32)	Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.
LastIndexOf(String)	Reports the zero-based index position of the last occurrence of a specified string within this instance.
LastIndexOf(Char)	Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance.
LastIndexOf(String, StringComparison)	Reports the zero-based index of the last occurrence of a specified string within the current String object. A parameter specifies the type of search to use for the specified string.

LastIndexOf(String, Int32, Int32, StringComparison)

Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for the specified number of character positions. A parameter specifies the type of comparison to perform when searching for the specified string.

C#

```
public int LastIndexOf (string value, int startIndex, int count,
StringComparison comparisonType);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

`count` Int32

The number of character positions to examine.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The zero-based starting index position of the `value` parameter if that string is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`count` is negative.

-or-

The current instance does not equal Empty, and `startIndex` is negative.

-or-

The current instance does not equal Empty, and `startIndex` is greater than the length of this instance.

-or-

The current instance does not equal Empty, and `startIndex` + 1 - `count` specifies a position that is not within this instance.

-or-

The current instance equals Empty and `start` is less than -1 or greater than zero.

-or-

The current instance equals Empty and `count` is greater than 1.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the LastIndexOf method that find the last occurrence of a string within another string using different values of the StringComparison enumeration.

C#

```
// This code example demonstrates the
// System.String.LastIndexOf(String, ..., StringComparison) methods.

using System;
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the last occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";
    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
```

```
                        Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                        CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
// the string was not found.
// Search using different values of StringComparsion. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, cat.Length,
sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the last occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire ca°t"

Part 1: Start index and count are specified.
```

```
Comparison: CurrentCulture                Location:  -1
Comparison: CurrentCultureIgnoreCase      Location:  12
Comparison: InvariantCulture              Location:  -1
Comparison: InvariantCultureIgnoreCase    Location:  12
Comparison: Ordinal                       Location:  -1
Comparison: OrdinalIgnoreCase             Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture                Location:  -1
Comparison: CurrentCultureIgnoreCase      Location:  12
Comparison: InvariantCulture              Location:  -1
Comparison: InvariantCultureIgnoreCase    Location:  12
Comparison: Ordinal                       Location:  -1
Comparison: OrdinalIgnoreCase             Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture                Location:  -1
Comparison: CurrentCultureIgnoreCase      Location:  12
Comparison: InvariantCulture              Location:  -1
Comparison: InvariantCultureIgnoreCase    Location:  12
Comparison: Ordinal                       Location:  -1
Comparison: OrdinalIgnoreCase             Location:  -1

*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The search begins at the `startIndex` character position and proceeds backward until either `value` is found or `count` character positions have been examined. For example, if `startIndex` is Length - 1, the method searches backward `count` characters from the last character in the string.

The `comparisonType` parameter specifies to search for the `value` parameter using:

- The current or invariant culture.
- A case-sensitive or case-insensitive search.
- Word or ordinal comparison rules.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `comparisonType` is not Ordinal or

OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed.

In the following example, the LastIndexOf(String, Int32, Int32, StringComparison) method is used to find the position of a soft hyphen (U+00AD) followed by an "m" in all but the first character position before the final "m" in two strings. Only one of the strings contains the required substring. If the example is run on .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string when it performs a culture-sensitive comparison. When it performs an ordinal comparison, however, it finds the substring only in the first string. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method returns the index of the "m" when it performs a culture-sensitive comparison. The method returns the index of the soft hyphen in the first string only when it performs an ordinal comparison.

C#

```
string searchString = "\u00ADm";

string s1 = "ani\u00ADmal";
string s2 = "animal";

int position;

position = s1.LastIndexOf('m');
if (position >= 1)
{
    Console.WriteLine(s1.LastIndexOf(searchString, position, position,
StringComparison.CurrentCulture));
    Console.WriteLine(s1.LastIndexOf(searchString, position, position,
StringComparison.Ordinal));
}

position = s2.LastIndexOf('m');
if (position >= 1)
{
    Console.WriteLine(s2.LastIndexOf(searchString, position, position,
StringComparison.CurrentCulture));
    Console.WriteLine(s2.LastIndexOf(searchString, position, position,
StringComparison.Ordinal));
}

// The example displays the following output:
//
// 4
// 3
// 3
// -1
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(String, Int32, StringComparison)

Reports the zero-based index of the last occurrence of a specified string within the current String object. The search starts at a specified character position and proceeds backward toward the beginning of the string. A parameter specifies the type of comparison to perform when searching for the specified string.

C#

```
public int LastIndexOf (string value, int startIndex, StringComparison
comparisonType);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The zero-based starting index position of the `value` parameter if that string is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

The current instance does not equal Empty, and `startIndex` is less than zero or greater than the length of the current instance.

-or-

The current instance equals Empty, and `startIndex` is less than -1 or greater than zero.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the LastIndexOf method that find the last occurrence of a string within another string using different values of the StringComparison enumeration.

C#

```
// This code example demonstrates the
// System.String.LastIndexOf(String, ..., StringComparison) methods.

using System;
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the last occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
```

```
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";
    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
                      Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                      CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
// the string was not found.
// Search using different values of StringComparsion. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, cat.Length,
sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
```

```
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the last occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire ca°t"

Part 1: Start index and count are specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The search begins at the `startIndex` character position and proceeds backward until either `value` is found or the first character position has been examined. For example, if `startIndex` is Length - 1, the method searches every character from the last character in the string to the beginning.

The `comparisonType` parameter specifies to search for the `value` parameter using the current or invariant culture, using a case-sensitive or case-insensitive search, and using word or ordinal comparison rules.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `comparisonType` is not Ordinal or OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed.

In the following example, the LastIndexOf(String, Int32, StringComparison) method is used to find the position of a soft hyphen (U+00AD) followed by an "m", starting with the final "m" in two strings. Only one of the strings contains the required substring. If the example is run on .NET Framework 4 or later, in both cases, because the soft hyphen is an ignorable character, the method returns the index of "m" in the string when it performs a culture-sensitive comparison. Note that in the case of the first string, which includes the soft hyphen followed by an "m", the method returns the index of the "m" and not the index of the soft hyphen. The method returns the index of the soft hyphen in the first string only when it performs an ordinal comparison.

C#

```
string searchString = "\u00ADm";

string s1 = "ani\u00ADmal";
string s2 = "animal";

int position;

position = s1.LastIndexOf('m');
if (position >= 0)
```

```
{
    Console.WriteLine(s1.LastIndexOf(searchString, position,
StringComparison.CurrentCulture));
    Console.WriteLine(s1.LastIndexOf(searchString, position,
StringComparison.Ordinal));
}

position = s2.LastIndexOf('m');
if (position >= 0)
{
    Console.WriteLine(s2.LastIndexOf(searchString, position,
StringComparison.CurrentCulture));
    Console.WriteLine(s2.LastIndexOf(searchString, position,
StringComparison.Ordinal));
}

// The example displays the following output:
//
// 4
// 3
// 3
// -1
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(String, Int32, Int32)

Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.

```
C#
```

```
public int LastIndexOf (string value, int startIndex, int count);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

`count` Int32

The number of character positions to examine.

Returns

Int32

The zero-based starting index position of `value` if that string is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

`count` is negative.

-or-

The current instance does not equal Empty, and `startIndex` is negative.

-or-

The current instance does not equal Empty, and `startIndex` is greater than the length of this instance.

-or-

The current instance does not equal Empty, and `startIndex` - `count` + 1 specifies a position that is not within this instance.

-or-

The current instance equals Empty and `start` is less than -1 or greater than zero.

-or-

The current instance equals Empty and `count` is greater than 1.

Examples

The following example finds the index of all occurrences of a string in substring, working from the end of the substring to the start of the substring.

C#

```
// Sample for String.LastIndexOf(String, Int32, Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    int count;
    int end;

    start = str.Length-1;
    end = start/2 - 1;
    Console.WriteLine("All occurrences of 'he' from position {0} to
{1}.", start, end);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("The string 'he' occurs at position(s): ");

    count = 0;
    at = 0;
    while((start > -1) && (at > -1))
        {
        count = start - end; //Count must be within the substring.
        at = str.LastIndexOf("he", start, count);
        if (at > -1)
            {
            Console.Write("{0} ", at);
            start = at - 1;
            }
```

```
        }
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
All occurrences of 'he' from position 66 to 32.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

The string 'he' occurs at position(s): 56 45
*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The search begins at the `startIndex` character position of this instance and proceeds backward toward the beginning until either `value` is found or `count` character positions have been examined. For example, if `startIndex` is Length - 1, the method searches backward `count` characters from the last character in the string.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed.

In the following example, the LastIndexOf method is used to find the position of a soft hyphen (U+00AD) followed by an "m" or "n" in two strings. Only one of the strings contains a soft hyphen. In the case of the string that includes the soft hyphen followed by an "m", `LastIndexOf` returns the index of the "m" when searching for the soft hyphen followed by "m".

C#

```
int position = 0;
string s1 = "ani\u00ADmal";
string s2 = "animal";

// Find the index of the soft hyphen followed by "n".
position = s1.LastIndexOf("m");
```

```
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s1.LastIndexOf("\u00ADn", position, position + 1));

position = s2.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s2.LastIndexOf("\u00ADn", position, position + 1));

// Find the index of the soft hyphen followed by "m".
position = s1.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s1.LastIndexOf("\u00ADm", position, position + 1));

position = s2.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s2.LastIndexOf("\u00ADm", position, position + 1));

// The example displays the following output:
//
// 'm' at position 4
// 1
// 'm' at position 3
// 1
// 'm' at position 4
// 4
// 'm' at position 3
// 3
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To perform this operation by using the comparison rules of the current culture, signal your intention explicitly by calling the LastIndexOf(String, Int32, Int32, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32

- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(Char, Int32, Int32)

Reports the zero-based index position of the last occurrence of the specified Unicode character in a substring within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.

C#

```
public int LastIndexOf (char value, int startIndex, int count);
```

Parameters

`value` Char

The Unicode character to seek.

`startIndex` Int32

The starting position of the search. The search proceeds from `startIndex` toward the beginning of this instance.

`count` Int32

The number of character positions to examine.

Returns

Int32

The zero-based index position of `value` if that character is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentOutOfRangeException

The current instance does not equal Empty, and `startIndex` is less than zero or greater than or equal to the length of this instance.

-or-

The current instance does not equal Empty, and `startIndex` - `count` + 1 is less than zero.

Examples

The following example finds the index of all occurrences of a character in a substring, working from the end of the substring to the start of the substring.

C#

```
// Sample for String.LastIndexOf(Char, Int32, Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    int count;
    int end;

    start = str.Length-1;
    end = start/2 - 1;
    Console.WriteLine("All occurrences of 't' from position {0} to {1}.",
start, end);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("The letter 't' occurs at position(s): ");
```

```
    count = 0;
    at = 0;
    while((start > -1) && (at > -1))
        {
        count = start - end; //Count must be within the substring.
        at = str.LastIndexOf('t', start, count);
        if (at > -1)
            {
            Console.Write("{0} ", at);
            start = at - 1;
            }
        }
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
All occurrences of 't' from position 66 to 32.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

The letter 't' occurs at position(s): 64 55 44 41 33


*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

This method begins searching at the `startIndex` character position and proceeds backward toward the beginning of this instance until either `value` is found or `count` character positions have been examined. For example, if `startIndex` is Length - 1, the method searches backward `count` characters from the last character in the string. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar value are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(String, Int32)

Reports the zero-based index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.

C#

```
public int LastIndexOf (string value, int startIndex);
```

Parameters

`value` String

The string to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

Returns

Int32

The zero-based starting index position of `value` if that string is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentOutOfRangeException

The current instance does not equal Empty, and `startIndex` is less than zero or greater than the length of the current instance.

-or-

The current instance equals Empty, and `startIndex` is less than -1 or greater than zero.

Examples

The following example finds the index of all occurrences of a string in target string, working from the end of the target string to the start of the target string.

C#

```
// Sample for String.LastIndexOf(String, Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;

    start = str.Length-1;
    Console.WriteLine("All occurrences of 'he' from position {0} to 0.",
start);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
```

```
    Console.Write("The string 'he' occurs at position(s): ");

    at = 0;
    while((start > -1) && (at > -1))
        {
        at = str.LastIndexOf("he", start);
        if (at > -1)
            {
            Console.Write("{0} ", at);
            start = at - 1;
            }
        }
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
All occurrences of 'he' from position 66 to 0.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

The string 'he' occurs at position(s): 56 45 8


*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The search begins at the `startIndex` character position of this instance and proceeds backward toward the beginning until either `value` is found or the first character position has been examined. For example, if `startIndex` is Length - 1, the method searches every character from the last character in the string to the beginning.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed. In the following example, the LastIndexOf(String, Int32) method is used to find a substring that includes a soft hyphen (U+00AD) and that precedes or includes the final "m" in a string. If the example is run on .NET Framework 4 or later, because the soft hyphen in the search

string is ignored, calling the method to find a substring that consists of the soft hyphen and "m" returns the position of the "m" in the string, whereas calling it to find a substring that consists of the soft hyphen and "n" returns the position of the "n".

C#

```
int position = 0;
string s1 = "ani\u00ADmal";
string s2 = "animal";

// Find the index of the soft hyphen followed by "n".
position = s1.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s1.LastIndexOf("\u00ADn", position));

position = s2.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s2.LastIndexOf("\u00ADn", position));

// Find the index of the soft hyphen followed by "m".
position = s1.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s1.LastIndexOf("\u00ADm", position));

position = s2.LastIndexOf("m");
Console.WriteLine($"'m' at position {position}");

if (position >= 0)
    Console.WriteLine(s2.LastIndexOf("\u00ADm", position));

// The example displays the following output:
//
// 'm' at position 4
// 1
// 'm' at position 3
// 1
// 'm' at position 4
// 4
// 'm' at position 3
// 3
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To find the index of a substring that precedes a particular character position by using the comparison rules of the current culture, signal your intention explicitly by calling the LastIndexOf(String, Int32, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(Char, Int32)

Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance. The search starts at a specified character position and proceeds backward toward the beginning of the string.

C#

```
public int LastIndexOf (char value, int startIndex);
```

Parameters

`value` Char

The Unicode character to seek.

`startIndex` Int32

The starting position of the search. The search proceeds from `startIndex` toward the beginning of this instance.

Returns

Int32

The zero-based index position of `value` if that character is found, or -1 if it is not found or if the current instance equals Empty.

Exceptions

ArgumentOutOfRangeException

The current instance does not equal Empty, and `startIndex` is less than zero or greater than or equal to the length of this instance.

Examples

The following example finds the index of all occurrences of a character in a string, working from the end of the string to the start of the string.

C#

```
// Sample for String.LastIndexOf(Char, Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;

    start = str.Length-1;
    Console.WriteLine("All occurrences of 't' from position {0} to 0.",
start);
```

```
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("The letter 't' occurs at position(s): ");

    at = 0;
    while((start > -1) && (at > -1))
        {
        at = str.LastIndexOf('t', start);
        if (at > -1)
            {
            Console.Write("{0} ", at);
            start = at - 1;
            }
        }
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
All occurrences of 't' from position 66 to 0.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

The letter 't' occurs at position(s): 64 55 44 41 33 11 7
*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1. This method begins searching at the `startIndex` character position of this instance and proceeds backward toward the beginning of the current instance until either `value` is found or the first character position has been examined. For example, if `startIndex` is Length - 1, the method searches every character from the last character in the string to the beginning. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(String)

Reports the zero-based index position of the last occurrence of a specified string within this instance.

C#

```
public int LastIndexOf (string value);
```

Parameters

`value` String

The string to seek.

Returns

Int32

The zero-based starting index position of `value` if that string is found, or -1 if it is not.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example removes opening and closing HTML tags from a string if the tags begin and end the string. If a string ends with a closing bracket character (">"), the example uses the LastIndexOf method to locate the start of the end tag.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] strSource = { "<b>This is bold text</b>", "<H1>This is
large Text</H1>",
                "<b><i><font color=green>This has multiple tags</font></i>
</b>",
                "<b>This has <i>embedded</i> tags.</b>",
                "This line ends with a greater than symbol and should not
be modified>" };

      // Strip HTML start and end tags from each string if they are
present.
      foreach (string s in strSource)
      {
         Console.WriteLine("Before: " + s);
         string item = s;
         // Use EndsWith to find a tag at the end of the line.
         if (item.Trim().EndsWith(">"))
         {
            // Locate the opening tag.
            int endTagStartPosition = item.LastIndexOf("</");
            // Remove the identified section, if it is valid.
            if (endTagStartPosition >= 0 )
               item = item.Substring(0, endTagStartPosition);

            // Use StartsWith to find the opening tag.
            if (item.Trim().StartsWith("<"))
            {
               // Locate the end of opening tab.
               int openTagEndPosition = item.IndexOf(">");
               // Remove the identified section, if it is valid.
               if (openTagEndPosition >= 0)
                  item = item.Substring(openTagEndPosition + 1);
            }
         }
```

```
            // Display the trimmed string.
            Console.WriteLine("After: " + item);
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//    Before: <b>This is bold text</b>
//    After: This is bold text
//
//    Before: <H1>This is large Text</H1>
//    After: This is large Text
//
//    Before: <b><i><font color=green>This has multiple tags</font></i>
</b>
//    After: <i><font color=green>This has multiple tags</font></i>
//
//    Before: <b>This has <i>embedded</i> tags.</b>
//    After: This has <i>embedded</i> tags.
//
//    Before: This line ends with a greater than symbol and should not be
modified>
//    After: This line ends with a greater than symbol and should not be
modified>
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The search begins at the last character position of this instance and proceeds backward toward the beginning until either `value` is found or the first character position has been examined.

This method performs a word (case-sensitive and culture-sensitive) search using the current culture.

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search, if `value` contains an ignorable character, the result is equivalent to searching with that character removed.

In the following example, the LastIndexOf(String) method is used to find two substrings (a soft hyphen followed by "n" and a soft hyphen followed by "m") in two strings. Only one of the strings contains a soft hyphen. If the example is run on .NET Framework 4 or later, in each case, because the soft hyphen is an ignorable

character, the result is the same as if the soft hyphen had not been included in `value`.

```
C#

string s1 = "ani\u00ADmal";
string s2 = "animal";

// Find the index of the last soft hyphen followed by "n".
Console.WriteLine(s1.LastIndexOf("\u00ADn"));
Console.WriteLine(s2.LastIndexOf("\u00ADn"));

// Find the index of the last soft hyphen followed by "m".
Console.WriteLine(s1.LastIndexOf("\u00ADm"));
Console.WriteLine(s2.LastIndexOf("\u00ADm"));

// The example displays the following output:
//
// 1
// 1
// 4
// 3
```

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To find the last index of a substring within a string instance by using the comparison rules of the current culture, signal your intention explicitly by calling the LastIndexOf(String, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- Int32
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(Char)

Reports the zero-based index position of the last occurrence of a specified Unicode character within this instance.

C#

```
public int LastIndexOf (char value);
```

Parameters

`value` Char

The Unicode character to seek.

Returns

Int32

The zero-based index position of `value` if that character is found, or -1 if it is not.

Examples

The following example defines an `ExtractFilename` method that uses the LastIndexOf(Char) method to find the last directory separator character in a string and to extract the string's file name. If the file exists, the method returns the file name without its path.

C#

```
using System;
using System.IO;
```

```
public class TestLastIndexOf
{
   public static void Main()
   {
      string filename;

      filename = ExtractFilename(@"C:\temp\");
      Console.WriteLine("{0}", String.IsNullOrEmpty(filename) ? "<none>"
: filename);

      filename = ExtractFilename(@"C:\temp\delegate.txt");
      Console.WriteLine("{0}", String.IsNullOrEmpty(filename) ? "<none>"
: filename);

      filename = ExtractFilename("delegate.txt");
      Console.WriteLine("{0}", String.IsNullOrEmpty(filename) ? "<none>"
: filename);

      filename = ExtractFilename(@"C:\temp\notafile.txt");
      Console.WriteLine("{0}", String.IsNullOrEmpty(filename) ? "<none>"
: filename);
   }

   public static string ExtractFilename(string filepath)
   {
      // If path ends with a "\", it's a path only so return
String.Empty.
      if (filepath.Trim().EndsWith(@"\"))
         return String.Empty;

      // Determine where last backslash is.
      int position = filepath.LastIndexOf('\\');
      // If there is no backslash, assume that this is a filename.
      if (position == -1)
      {
         // Determine whether file exists in the current directory.
         if (File.Exists(Environment.CurrentDirectory +
Path.DirectorySeparatorChar + filepath))
            return filepath;
         else
            return String.Empty;
      }
      else
      {
         // Determine whether file exists using filepath.
         if (File.Exists(filepath))
            // Return filename without file path.
            return filepath.Substring(position + 1);
         else
            return String.Empty;
      }
   }
}
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

This method begins searching at the last character position of this instance and proceeds backward toward the beginning until either `value` is found or the first character position has been examined. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOfAny(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOf(String, StringComparison)

Reports the zero-based index of the last occurrence of a specified string within the current String object. A parameter specifies the type of search to use for the specified string.

```
C#

public int LastIndexOf (string value, StringComparison comparisonType);
```

Parameters

`value` String

The string to seek.

`comparisonType` StringComparison

One of the enumeration values that specifies the rules for the search.

Returns

Int32

The zero-based starting index position of the `value` parameter if that string is found, or -1 if it is not.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`comparisonType` is not a valid StringComparison value.

Examples

The following example demonstrates three overloads of the LastIndexOf method that find the last occurrence of a string within another string using different values of the StringComparison enumeration.

```
C#

// This code example demonstrates the
// System.String.LastIndexOf(String, ..., StringComparison) methods.

using System;
```

```
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string intro = "Find the last occurrence of a character using
different " +
                   "values of StringComparison.";
    string resultFmt = "Comparison: {0,-28} Location: {1,3}";

// Define a string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
    string CapitalAWithRing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string cat = "A Cheshire c" + "\u0061\u030a" + "t";
    int loc = 0;
    StringComparison[] scValues = {
        StringComparison.CurrentCulture,
        StringComparison.CurrentCultureIgnoreCase,
        StringComparison.InvariantCulture,
        StringComparison.InvariantCultureIgnoreCase,
        StringComparison.Ordinal,
        StringComparison.OrdinalIgnoreCase };

// Clear the screen and display an introduction.
    Console.Clear();
    Console.WriteLine(intro);

// Display the current culture because culture affects the result. For
example,
// try this code example with the "sv-SE" (Swedish-Sweden) culture.

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Console.WriteLine("The current culture is \"{0}\" - {1}.",
                      Thread.CurrentThread.CurrentCulture.Name,
                      Thread.CurrentThread.CurrentCulture.DisplayName);

// Display the string to search for and the string to search.
    Console.WriteLine("Search for the string \"{0}\" in the string \"
{1}\"",
                      CapitalAWithRing, cat);
    Console.WriteLine();

// Note that in each of the following searches, we look for
// LATIN CAPITAL LETTER A WITH RING ABOVE in a string that contains
// LATIN SMALL LETTER A WITH RING ABOVE. A result value of -1 indicates
// the string was not found.
```

```
// Search using different values of StringComparsion. Specify the start
// index and count.

    Console.WriteLine("Part 1: Start index and count are specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, cat.Length,
sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion. Specify the
// start index.
    Console.WriteLine("\nPart 2: Start index is specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, cat.Length-1, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }

// Search using different values of StringComparsion.
    Console.WriteLine("\nPart 3: Neither start index nor count is
specified.");
    foreach (StringComparison sc in scValues)
        {
        loc = cat.LastIndexOf(CapitalAWithRing, sc);
        Console.WriteLine(resultFmt, sc, loc);
        }
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

This code example produces the following results:

Find the last occurrence of a character using different values of
StringComparison.
The current culture is "en-US" - English (United States).
Search for the string "Å" in the string "A Cheshire ca°t"

Part 1: Start index and count are specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 2: Start index is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
```

```
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

Part 3: Neither start index nor count is specified.
Comparison: CurrentCulture               Location:  -1
Comparison: CurrentCultureIgnoreCase     Location:  12
Comparison: InvariantCulture             Location:  -1
Comparison: InvariantCultureIgnoreCase   Location:  12
Comparison: Ordinal                      Location:  -1
Comparison: OrdinalIgnoreCase            Location:  -1

*/
```

Remarks

Index numbering starts from zero. That is, the first character in the string is at index zero and the last is at Length - 1.

The `comparisonType` parameter specifies to search for the `value` parameter using:

- The current or invariant culture.
- A case-sensitive or case-insensitive search.
- Word or ordinal comparison rules.

The search begins at the last character position of this instance and proceeds backward toward the beginning until either `value` is found or the first character position has been examined.

Notes to Callers

Character sets include ignorable characters, which are characters that are not considered when performing a linguistic or culture-sensitive comparison. In a culture-sensitive search (that is, if `options` is not Ordinal or OrdinalIgnoreCase), if `value` contains an ignorable character, the result is equivalent to searching with that character removed.

In the following example, the LastIndexOf(String, StringComparison) method is used to find two substrings (a soft hyphen followed by "n", and a soft hyphen followed by "m") in two strings. Only one of the strings contains a soft hyphen. If the example is run on .NET Framework 4 or later, because the soft hyphen is an ignorable character, a culture-sensitive search returns the same value that it would return if the soft hyphen were not included in the search string. An ordinal search, however, successfully finds the soft hyphen in one string and reports that it is absent from the second string.

C#

```
string s1 = "ani\u00ADmal";
string s2 = "animal";

Console.WriteLine("Culture-sensitive comparison:");

// Use culture-sensitive comparison to find the last soft hyphen followed
by "n".
Console.WriteLine(s1.LastIndexOf("\u00ADn",
StringComparison.CurrentCulture));
Console.WriteLine(s2.LastIndexOf("\u00ADn",
StringComparison.CurrentCulture));

// Use culture-sensitive comparison to find the last soft hyphen followed
by "m".
Console.WriteLine(s1.LastIndexOf("\u00ADm",
StringComparison.CurrentCulture));
Console.WriteLine(s2.LastIndexOf("\u00ADm",
StringComparison.CurrentCulture));

Console.WriteLine("Ordinal comparison:");

// Use ordinal comparison to find the last soft hyphen followed by "n".
Console.WriteLine(s1.LastIndexOf("\u00ADn", StringComparison.Ordinal));
Console.WriteLine(s2.LastIndexOf("\u00ADn", StringComparison.Ordinal));

// Use ordinal comparison to find the last soft hyphen followed by "m".
Console.WriteLine(s1.LastIndexOf("\u00ADm", StringComparison.Ordinal));
Console.WriteLine(s2.LastIndexOf("\u00ADm", StringComparison.Ordinal));

// The example displays the following output:
//
// Culture-sensitive comparison:
// 1
// 1
// 4
// 3
// Ordinal comparison:
// -1
// -1
// 3
// -1
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.LastIndexOfAny Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The method returns -1 if the characters in the array are not found in this instance.

Overloads

Expand table

LastIndexOfAny(Char[])	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.
LastIndexOfAny(Char[], Int32)	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string.
LastIndexOfAny(Char[], Int32, Int32)	Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.

LastIndexOfAny(Char[])

Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.

C#

```
public int LastIndexOfAny (char[] anyOf);
```

Parameters

anyOf Char[]

A Unicode character array containing one or more characters to seek.

Returns

Int32

The index position of the last occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found.

Exceptions

ArgumentNullException

`anyOf` is `null`.

Examples

The following example finds the index of the last occurrence of any character in the string "is" within another string.

C#

```
// Sample for String.LastIndexOfAny(Char[])
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    string target = "is";
    char[] anyOf = target.ToCharArray();

    start = str.Length-1;
    Console.WriteLine("The last character occurrence  from position {0}
to 0.", start);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("A character in '{0}' occurs at position: ", target);

    at = str.LastIndexOfAny(anyOf);
    if (at > -1)
```

```
        Console.Write(at);
    else
        Console.Write("(not found)");
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
The last character occurrence  from position 66 to 0.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

A character in 'is' occurs at position: 58


*/
```

Remarks

Index numbering starts from zero.

This method begins searching at the last character position of this instance and proceeds backward toward the beginning until either a character in `anyOf` is found or the first character position has been examined. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- Array
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOf(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOfAny(Char[], Int32)

Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string.

C#

```
public int LastIndexOfAny (char[] anyOf, int startIndex);
```

Parameters

`anyOf` Char[]

A Unicode character array containing one or more characters to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

Returns

Int32

The index position of the last occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`anyOf` is `null`.

ArgumentOutOfRangeException

The current instance does not equal Empty, and `startIndex` specifies a position that is not within this instance.

Examples

The following example finds the index of the last occurrence of any character in the string "is" within a substring of another string.

C#

```
// Sample for String.LastIndexOfAny(Char[], Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    string target = "is";
    char[] anyOf = target.ToCharArray();

    start = (str.Length-1)/2;
    Console.WriteLine("The last character occurrence  from position {0}
to 0.", start);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("A character in '{0}' occurs at position: ", target);

    at = str.LastIndexOfAny(anyOf, start);
    if (at > -1)
        Console.Write(at);
    else
        Console.Write("(not found)");
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
}
/*
This example produces the following results:
The last character occurrence  from position 33 to 0.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
```

```
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

A character in 'is' occurs at position: 12


*/
```

Remarks

Index numbering starts from zero.

This method begins searching at the `startIndex` character position of this instance and proceeds backward toward the beginning until either a character in `anyOf` is found or the first character position has been examined. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- Array
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOf(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

LastIndexOfAny(Char[], Int32, Int32)

Reports the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and proceeds backward toward the beginning of the string for a specified number of character positions.

C#

```
public int LastIndexOfAny (char[] anyOf, int startIndex, int count);
```

Parameters

`anyOf` Char[]

A Unicode character array containing one or more characters to seek.

`startIndex` Int32

The search starting position. The search proceeds from `startIndex` toward the beginning of this instance.

`count` Int32

The number of character positions to examine.

Returns

Int32

The index position of the last occurrence in this instance where any character in `anyOf` was found; -1 if no character in `anyOf` was found or if the current instance equals Empty.

Exceptions

ArgumentNullException

`anyOf` is `null`.

ArgumentOutOfRangeException

The current instance does not equal Empty, and `count` or `startIndex` is negative.

-or-

The current instance does not equal Empty, and `startIndex` minus `count` + 1 is less than zero.

Examples

The following example finds the index of the last occurrence of any character in the string "aid" within a substring of another string.

C#

```
// Sample for String.LastIndexOfAny(Char[], Int32, Int32)
using System;

class Sample {
    public static void Main() {

    string br1 = "0----+----1----+----2----+----3----+----4----+----5----
+----6----+-";
    string br2 =
"0123456789012345678901234567890123456789012345678901234567890123456";
    string str = "Now is the time for all good men to come to the aid of
their party.";
    int start;
    int at;
    int count;
    string target = "aid";
    char[] anyOf = target.ToCharArray();

    start = ((str.Length-1)*2)/3;
    count = (str.Length-1)/3;
    Console.WriteLine("The last character occurrence from position {0}
for {1} characters.", start, count);
    Console.WriteLine("{1}{0}{2}{0}{3}{0}", Environment.NewLine, br1,
br2, str);
    Console.Write("A character in '{0}' occurs at position: ", target);

    at = str.LastIndexOfAny(anyOf, start, count);
    if (at > -1)
        Console.Write(at);
    else
        Console.Write("(not found)");
    Console.Write("{0}{0}{0}", Environment.NewLine);
    }
```

```
}
/*
This example produces the following results:
The last character occurrence from position 44 for 22 characters.
0----+----1----+----2----+----3----+----4----+----5----+----6----+-
0123456789012345678901234567890123456789012345678901234567890123456
Now is the time for all good men to come to the aid of their party.

A character in 'aid' occurs at position: 27
*/
```

Remarks

Index numbering starts from zero.

This method begins searching at the `startIndex` character position of this instance and proceeds backward toward the beginning until either a character in `anyOf` is found or `count` character positions have been examined. The search is case-sensitive.

This method performs an ordinal (culture-insensitive) search, where a character is considered equivalent to another character only if their Unicode scalar values are the same. To perform a culture-sensitive search, use the CompareInfo.LastIndexOf method, where a Unicode scalar value representing a precomposed character, such as the ligature "Æ" (U+00C6), might be considered equivalent to any occurrence of the character's components in the correct sequence, such as "AE" (U+0041, U+0045), depending on the culture.

See also

- Char
- Int32
- Array
- IndexOf(Char)
- IndexOfAny(Char[])
- LastIndexOf(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Normalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string whose binary representation is in a particular Unicode normalization form.

Overloads

Expand table

Normalize(NormalizationForm)	Returns a new string whose textual value is the same as this string, but whose binary representation is in the specified Unicode normalization form.
Normalize()	Returns a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.

Examples

The following example normalizes a string to each of four normalization forms, confirms the string was normalized to the specified normalization form, then lists the code points in the normalized string.

C#

```
using System;
using System.Text;

class Example
{
    public static void Main()
    {
       // Character c; combining characters acute and cedilla; character 3/4
       string s1 = new String( new char[] {'\u0063', '\u0301', '\u0327',
'\u00BE'});
       string s2 = null;
       string divider = new String('-', 80);
```

```
        divider = String.Concat(Environment.NewLine, divider,
Environment.NewLine);

        Show("s1", s1);
        Console.WriteLine();
        Console.WriteLine("U+0063 = LATIN SMALL LETTER C");
        Console.WriteLine("U+0301 = COMBINING ACUTE ACCENT");
        Console.WriteLine("U+0327 = COMBINING CEDILLA");
        Console.WriteLine("U+00BE = VULGAR FRACTION THREE QUARTERS");
        Console.WriteLine(divider);

        Console.WriteLine("A1) Is s1 normalized to the default form (Form
C)?: {0}",
                                    s1.IsNormalized());
        Console.WriteLine("A2) Is s1 normalized to Form C?:  {0}",

s1.IsNormalized(NormalizationForm.FormC));
        Console.WriteLine("A3) Is s1 normalized to Form D?:  {0}",

s1.IsNormalized(NormalizationForm.FormD));
        Console.WriteLine("A4) Is s1 normalized to Form KC?: {0}",

s1.IsNormalized(NormalizationForm.FormKC));
        Console.WriteLine("A5) Is s1 normalized to Form KD?: {0}",

s1.IsNormalized(NormalizationForm.FormKD));

        Console.WriteLine(divider);

        Console.WriteLine("Set string s2 to each normalized form of string
s1.");
        Console.WriteLine();
        Console.WriteLine("U+1E09 = LATIN SMALL LETTER C WITH CEDILLA AND
ACUTE");
        Console.WriteLine("U+0033 = DIGIT THREE");
        Console.WriteLine("U+2044 = FRACTION SLASH");
        Console.WriteLine("U+0034 = DIGIT FOUR");
        Console.WriteLine(divider);

        s2 = s1.Normalize();
        Console.Write("B1) Is s2 normalized to the default form (Form C)?:
");
        Console.WriteLine(s2.IsNormalized());
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormC);
        Console.Write("B2) Is s2 normalized to Form C?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormC));
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormD);
        Console.Write("B3) Is s2 normalized to Form D?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormD));
```

```
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormKC);
        Console.Write("B4) Is s2 normalized to Form KC?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormKC));
        Show("s2", s2);
        Console.WriteLine();

        s2 = s1.Normalize(NormalizationForm.FormKD);
        Console.Write("B5) Is s2 normalized to Form KD?: ");
        Console.WriteLine(s2.IsNormalized(NormalizationForm.FormKD));
        Show("s2", s2);
        Console.WriteLine();
    }

    private static void Show(string title, string s)
    {
        Console.Write("Characters in string {0} = ", title);
        foreach(short x in s) {
            Console.Write("{0:X4} ", x);
        }
        Console.WriteLine();
    }
}
/*
This example produces the following results:

Characters in string s1 = 0063 0301 0327 00BE

U+0063 = LATIN SMALL LETTER C
U+0301 = COMBINING ACUTE ACCENT
U+0327 = COMBINING CEDILLA
U+00BE = VULGAR FRACTION THREE QUARTERS

--------------------------------------------------------------------------
----

A1) Is s1 normalized to the default form (Form C)?: False
A2) Is s1 normalized to Form C?:  False
A3) Is s1 normalized to Form D?:  False
A4) Is s1 normalized to Form KC?: False
A5) Is s1 normalized to Form KD?: False

--------------------------------------------------------------------------
----

Set string s2 to each normalized form of string s1.

U+1E09 = LATIN SMALL LETTER C WITH CEDILLA AND ACUTE
U+0033 = DIGIT THREE
U+2044 = FRACTION SLASH
U+0034 = DIGIT FOUR

--------------------------------------------------------------------------
```

```
----

B1) Is s2 normalized to the default form (Form C)?: True
Characters in string s2 = 1E09 00BE

B2) Is s2 normalized to Form C?: True
Characters in string s2 = 1E09 00BE

B3) Is s2 normalized to Form D?: True
Characters in string s2 = 0063 0327 0301 00BE

B4) Is s2 normalized to Form KC?: True
Characters in string s2 = 1E09 0033 2044 0034

B5) Is s2 normalized to Form KD?: True
Characters in string s2 = 0063 0327 0301 0033 2044 0034

*/
```

Normalize(NormalizationForm)

Returns a new string whose textual value is the same as this string, but whose binary representation is in the specified Unicode normalization form.

```
C#

public string Normalize (System.Text.NormalizationForm
normalizationForm);
```

Parameters

`normalizationForm` NormalizationForm

A Unicode normalization form.

Returns

String

A new string whose textual value is the same as this string, but whose binary representation is in the normalization form specified by the `normalizationForm` parameter.

Exceptions

ArgumentException

The current instance contains invalid Unicode characters.

Remarks

Some Unicode characters have multiple equivalent binary representations consisting of sets of combining and/or composite Unicode characters. The existence of multiple representations for a single character complicates searching, sorting, matching, and other operations.

The Unicode standard defines a process called normalization that returns one binary representation when given any of the equivalent binary representations of a character. Normalization can be performed with several algorithms, called normalization forms, that obey different rules. .NET supports the four normalization forms (C, D, KC, and KD) that are defined by the Unicode standard. When two strings are represented in the same normalization form, they can be compared by using ordinal comparison.

To normalize and compare two strings, do the following:

1. Obtain the strings to be compared from an input source, such as a file or a user input device.

2. Call the Normalize(NormalizationForm) method to normalize the strings to a specified normalization form.

3. To compare two strings, call a method that supports ordinal string comparison, such as the Compare(String, String, StringComparison) method, and supply a value of StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase as the StringComparison argument. To sort an array of normalized strings, pass a `comparer` value of StringComparer.Ordinal or StringComparer.OrdinalIgnoreCase to an appropriate overload of Array.Sort.

4. Emit the strings in the sorted output based on the order indicated by the previous step.

For a description of supported Unicode normalization forms, see System.Text.NormalizationForm.

Notes to Callers

The IsNormalized method returns `false` as soon as it encounters the first non-normalized character in a string. Therefore, if a string contains non-normalized

characters followed by invalid Unicode characters, the Normalize method may throw an ArgumentException although IsNormalized returns `false`.

See also

- IsNormalized()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Normalize()

Returns a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.

C#

```
public string Normalize ();
```

Returns

String

A new, normalized string whose textual value is the same as this string, but whose binary representation is in normalization form C.

Exceptions

ArgumentException

The current instance contains invalid Unicode characters.

Remarks

Some Unicode characters have multiple equivalent binary representations consisting of sets of combining and/or composite Unicode characters. For example, any of the following code points can represent the letter "ắ":

- U+1EAF
- U+0103 U+0301
- U+0061 U+0306 U+0301

The existence of multiple representations for a single character complicates searching, sorting, matching, and other operations.

The Unicode standard defines a process called normalization that returns one binary representation when given any of the equivalent binary representations of a character. Normalization can be performed with several algorithms, called normalization forms, that obey different rules. .NET supports the four normalization forms (C, D, KC, and KD) that are defined by the Unicode standard. When two strings are represented in the same normalization form, they can be compared by using ordinal comparison.

To normalize and compare two strings, do the following:

1. Obtain the strings to be compared from an input source, such as a file or a user input device.
2. Call the Normalize() method to normalize the strings to normalization form C.
3. To compare two strings, call a method that supports ordinal string comparison, such as the Compare(String, String, StringComparison) method, and supply a value of StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase as the StringComparison argument. To sort an array of normalized strings, pass a `comparer` value of StringComparer.Ordinal or StringComparer.OrdinalIgnoreCase to an appropriate overload of Array.Sort.
4. Emit the strings in the sorted output based on the order indicated by the previous step.

For a description of supported Unicode normalization forms, see System.Text.NormalizationForm.

Notes to Callers

The IsNormalized method returns `false` as soon as it encounters the first non-normalized character in a string. Therefore, if a string contains non-normalized characters followed by invalid Unicode characters, the Normalize method will throw an ArgumentException although IsNormalized returns `false`.

See also

- IsNormalized()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.PadLeft Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string of a specified length in which the beginning of the current string is padded with spaces or with a specified Unicode character.

Overloads

Expand table

PadLeft(Int32)	Returns a new string that right-aligns the characters in this instance by padding them with spaces on the left, for a specified total length.
PadLeft(Int32, Char)	Returns a new string that right-aligns the characters in this instance by padding them on the left with a specified Unicode character, for a specified total length.

PadLeft(Int32)

Returns a new string that right-aligns the characters in this instance by padding them with spaces on the left, for a specified total length.

C#

```
public string PadLeft (int totalWidth);
```

Parameters

`totalWidth` Int32

The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

Returns

String

A new string that is equivalent to this instance, but right-aligned and padded on the left with as many spaces as needed to create a length of `totalWidth`. However, if `totalWidth` is less than the length of this instance, the method returns a reference to the existing instance. If `totalWidth` is equal to the length of this instance, the method returns a new string that is identical to this instance.

Exceptions

ArgumentOutOfRangeException

`totalWidth` is less than zero.

Examples

The following example demonstrates the PadLeft method.

C#

```
string str = "BBQ and Slaw";
Console.WriteLine(str.PadLeft(15));  // Displays "   BBQ and Slaw".
Console.WriteLine(str.PadLeft(5));   // Displays "BBQ and Slaw".
```

Remarks

A Unicode space is defined as hexadecimal 0x0020.

The PadLeft(Int32) method pads the beginning of the returned string. This means that, when used with right-to-left languages, it pads the right portion of the string.

> **Note**
>
> If the **PadLeft** method pads the current instance with white-space characters, this method does not modify the value of the current instance. Instead, it returns a new string that is padded with leading white space so that its total length is `totalWidth` characters.

See also

- Int32
- PadRight(Int32)

- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

PadLeft(Int32, Char)

Returns a new string that right-aligns the characters in this instance by padding them on the left with a specified Unicode character, for a specified total length.

C#

```
public string PadLeft (int totalWidth, char paddingChar);
```

Parameters

`totalWidth` Int32

The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

`paddingChar` Char

A Unicode padding character.

Returns

String

A new string that is equivalent to this instance, but right-aligned and padded on the left with as many `paddingChar` characters as needed to create a length of `totalWidth`. However, if `totalWidth` is less than the length of this instance, the

method returns a reference to the existing instance. If `totalWidth` is equal to the length of this instance, the method returns a new string that is identical to this instance.

Exceptions

ArgumentOutOfRangeException

`totalWidth` is less than zero.

Examples

The following example demonstrates the PadLeft method.

C#

```
using System;

class Sample
{
   public static void Main()
   {
   string str = "forty-two";
   char pad = '.';

   Console.WriteLine(str.PadLeft(15, pad));
   Console.WriteLine(str.PadLeft(2, pad));
   }
}
// The example displays the following output:
//       ......forty-two
//       forty-two
```

Remarks

The PadLeft(Int32, Char) method pads the beginning of the returned string. This means that, when used with right-to-left languages, it pads the right portion of the string.

> **Note**
>
> If the **PadLeft** method pads the current instance with white-space characters, this method does not modify the value of the current instance. Instead, it returns a new string that is padded with leading `paddingChar` characters so that its total length is `totalWidth` characters.

See also

- Char
- Int32
- PadRight(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.PadRight Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string of a specified length in which the end of the current string is padded with spaces or with a specified Unicode character.

Overloads

Expand table

PadRight(Int32)	Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.
PadRight(Int32, Char)	Returns a new string that left-aligns the characters in this string by padding them on the right with a specified Unicode character, for a specified total length.

PadRight(Int32)

Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.

C#

```
public string PadRight (int totalWidth);
```

Parameters

`totalWidth` Int32

The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

Returns

String

A new string that is equivalent to this instance, but left-aligned and padded on the right with as many spaces as needed to create a length of `totalWidth`. However, if `totalWidth` is less than the length of this instance, the method returns a reference to the existing instance. If `totalWidth` is equal to the length of this instance, the method returns a new string that is identical to this instance.

Exceptions

ArgumentOutOfRangeException

`totalWidth` is less than zero.

Examples

The following example demonstrates the PadRight method.

C#

```
string str;
str = "BBQ and Slaw";

Console.Write("|");
Console.Write(str.PadRight(15));
Console.WriteLine("|");       // Displays "|BBQ and Slaw   |".

Console.Write("|");
Console.Write(str.PadRight(5));
Console.WriteLine("|");       // Displays "|BBQ and Slaw|".
```

Remarks

A Unicode space is defined as hexadecimal 0x0020.

The PadRight(Int32) method pads the end of the returned string. This means that, when used with right-to-left languages, it pads the left portion of the string.

> **Note**
>
> If the **PadRight** method pads the current instance with white-space characters, this method does not modify the value of the current instance. Instead, it returns a new string that is padded with trailing white space so that its total length is `totalWidth` characters.

See also

- Int32
- PadLeft(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

PadRight(Int32, Char)

Returns a new string that left-aligns the characters in this string by padding them on the right with a specified Unicode character, for a specified total length.

C#

```
public string PadRight (int totalWidth, char paddingChar);
```

Parameters

`totalWidth` Int32

The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

`paddingChar` Char

A Unicode padding character.

Returns

String

A new string that is equivalent to this instance, but left-aligned and padded on the right with as many `paddingChar` characters as needed to create a length of `totalWidth`. However, if `totalWidth` is less than the length of this instance, the method returns a reference to the existing instance. If `totalWidth` is equal to the length of this instance, the method returns a new string that is identical to this instance.

Exceptions

ArgumentOutOfRangeException

`totalWidth` is less than zero.

Examples

The following example demonstrates the PadRight method.

C#

```
string str = "forty-two";
char pad = '.';

Console.WriteLine(str.PadRight(15, pad));    // Displays "forty-
two......".
Console.WriteLine(str.PadRight(2,  pad));    // Displays "forty-two".
```

Remarks

The PadRight(Int32, Char) method pads the end of the returned string. This means that, when used with right-to-left languages, it pads the left portion of the string.

> **Note**
>
> If the **PadRight** method pads the current instance with white-space characters, this method does not modify the value of the current instance. Instead, it returns a new string that is padded with trailing `paddingChar` characters so that its total length is `totalWidth` characters.

See also

- Char
- Int32

- PadLeft(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Remove Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string in which a specified number of characters from the current string are deleted.

Overloads

Expand table

Remove(Int32)	Returns a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
Remove(Int32, Int32)	Returns a new string in which a specified number of characters in the current instance beginning at a specified position have been deleted.

Remove(Int32)

Returns a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.

C#

```
public string Remove (int startIndex);
```

Parameters

`startIndex` Int32

The zero-based position to begin deleting characters.

Returns

String

A new string that is equivalent to this string except for the removed characters.

Exceptions

ArgumentOutOfRangeException

`startIndex` is less than zero.

-or-

`startIndex` specifies a position that is not within this string.

Examples

The following example demonstrates the Remove method. The next-to-last case removes all text starting from the specified index through the end of the string. The last case removes three characters starting from the specified index.

C#

```
// This example demonstrates the String.Remove() method.
using System;

class Sample
{
    public static void Main()
    {
        string s = "abc---def";

        Console.WriteLine("Index: 012345678");
        Console.WriteLine("1)     {0}", s);
        Console.WriteLine("2)     {0}", s.Remove(3));
        Console.WriteLine("3)     {0}", s.Remove(3, 3));
    }
}
/*
This example produces the following results:

Index: 012345678
1)     abc---def
2)     abc
3)     abcdef

*/
```

Remarks

In the .NET Framework, strings are zero-based. The value of the `startIndex` parameter can range from zero to one less than the length of the string instance.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters from position `startIndex` to the end of the original string have been removed.

See also

- Int32
- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Replace(Char, Char)
- Split(Char[])
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Remove(Int32, Int32)

Returns a new string in which a specified number of characters in the current instance beginning at a specified position have been deleted.

```C#
public string Remove (int startIndex, int count);
```

Parameters

`startIndex` Int32

The zero-based position to begin deleting characters.

`count` Int32

The number of characters to delete.

Returns

String

A new string that is equivalent to this instance except for the removed characters.

Exceptions

ArgumentOutOfRangeException

Either `startIndex` or `count` is less than zero.

-or-

`startIndex` plus `count` specify a position outside this instance.

Examples

The following example demonstrates how you can remove the middle name from a complete name.

```C#
using System;

public class RemoveTest
{
    public static void Main()
    {

        string name = "Michelle Violet Banks";

        Console.WriteLine("The entire name is '{0}'", name);
```

```
        // Remove the middle name, identified by finding the spaces in
the name.
        int foundS1 = name.IndexOf(" ");
        int foundS2 = name.IndexOf(" ", foundS1 + 1);

        if (foundS1 != foundS2 && foundS1 >= 0)
        {
            name = name.Remove(foundS1 + 1, foundS2 - foundS1);

            Console.WriteLine("After removing the middle name, we are
left with '{0}'", name);
        }
    }
}
// The example displays the following output:
//       The entire name is 'Michelle Violet Banks'
//       After removing the middle name, we are left with 'Michelle
Banks'
```

Remarks

In the .NET Framework, strings are zero-based. The value of the `startIndex` parameter can range from zero to one less than the length of the string instance.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which the number of characters specified by the `count` parameter have been removed. The characters are removed at the position specified by `startIndex`.

See also

- Int32
- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Replace(Char, Char)
- Split(Char[])
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Replace Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string in which all occurrences of a specified Unicode character or String in the current string are replaced with another specified Unicode character or String.

Overloads

Expand table

Replace(Char, Char)	Returns a new string in which all occurrences of a specified Unicode character in this instance are replaced with another specified Unicode character.
Replace(String, String)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
Replace(String, String, StringComparison)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided comparison type.
Replace(String, String, Boolean, CultureInfo)	Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided culture and case sensitivity.

Replace(Char, Char)

Returns a new string in which all occurrences of a specified Unicode character in this instance are replaced with another specified Unicode character.

C#

```
public string Replace (char oldChar, char newChar);
```

Parameters

`oldChar` Char

The Unicode character to be replaced.

`newChar` Char

The Unicode character to replace all occurrences of `oldChar`.

Returns

String

A string that is equivalent to this instance except that all instances of `oldChar` are replaced with `newChar`. If `oldChar` is not found in the current instance, the method returns the current instance unchanged.

Examples

The following example creates a comma separated value list by substituting commas for the blanks between a series of numbers.

C#

```
String str = "1 2 3 4 5 6 7 8 9";
Console.WriteLine("Original string: \"{0}\"", str);
Console.WriteLine("CSV string:      \"{0}\"", str.Replace(' ', ','));

// This example produces the following output:
// Original string: "1 2 3 4 5 6 7 8 9"
// CSV string:      "1,2,3,4,5,6,7,8,9"
```

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) search to find `oldChar`.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all occurrences of `oldChar` are replaced by `newChar`.

Because this method returns the modified string, you can chain together successive calls to the Replace method to perform multiple replacements on the original string.

Method calls are executed from left to right. The following example provides an illustration.

```C#
String s = new String('a', 3);
Console.WriteLine("The initial string: '{0}'", s);
s = s.Replace('a', 'b').Replace('b', 'c').Replace('c', 'd');
Console.WriteLine("The final string: '{0}'", s);

// The example displays the following output:
//       The initial string: 'aaa'
//       The final string: 'ddd'
```

See also

- Char
- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Remove(Int32, Int32)
- Split(Char[])
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Replace(String, String)

Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.

```
C#

public string Replace (string oldValue, string? newValue);
```

Parameters

`oldValue` String

The string to be replaced.

`newValue` String

The string to replace all occurrences of `oldValue`.

Returns

String

A string that is equivalent to the current string except that all instances of `oldValue` are replaced with `newValue`. If `oldValue` is not found in the current instance, the method returns the current instance unchanged.

Exceptions

ArgumentNullException

`oldValue` is `null`.

ArgumentException

`oldValue` is the empty string ("").

Examples

The following example demonstrates how you can use the Replace method to correct a spelling error.

```
C#

string errString = "This docment uses 3 other docments to docment the
docmentation";

Console.WriteLine("The original string is:{0}'{1}'{0}",
Environment.NewLine, errString);
```

```
// Correct the spelling of "document".

string correctString = errString.Replace("docment", "document");

Console.WriteLine("After correcting the string, the result is:{0}'{1}'",
        Environment.NewLine, correctString);

// This code example produces the following output:
//
// The original string is:
// 'This docment uses 3 other docments to docment the docmentation'
//
// After correcting the string, the result is:
// 'This document uses 3 other documents to document the documentation'
//
```

Remarks

If `newValue` is `null`, all occurrences of `oldValue` are removed.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all occurrences of `oldValue` are replaced by `newValue`.

This method performs an ordinal (case-sensitive and culture-insensitive) search to find `oldValue`.

Because this method returns the modified string, you can chain together successive calls to the Replace method to perform multiple replacements on the original string. Method calls are executed from left to right. The following example provides an illustration.

C#

```
String s = "aaa";
Console.WriteLine("The initial string: '{0}'", s);
s = s.Replace("a", "b").Replace("b", "c").Replace("c", "d");
Console.WriteLine("The final string: '{0}'", s);

// The example displays the following output:
//       The initial string: 'aaa'
//       The final string: 'ddd'
```

See also

- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Remove(Int32, Int32)
- Split(Char[])
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Replace(String, String, StringComparison)

Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided comparison type.

C#

```
public string Replace (string oldValue, string? newValue,
StringComparison comparisonType);
```

Parameters

`oldValue` String

The string to be replaced.

`newValue` String

The string to replace all occurrences of `oldValue`.

`comparisonType` StringComparison

One of the enumeration values that determines how `oldValue` is searched within this instance.

Returns

String

A string that is equivalent to the current string except that all instances of `oldValue` are replaced with `newValue`. If `oldValue` is not found in the current instance, the method returns the current instance unchanged.

Exceptions

ArgumentNullException

`oldValue` is `null`.

ArgumentException

`oldValue` is the empty string ("").

Remarks

If `newValue` is `null`, all occurrences of `oldValue` are removed.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all occurrences of `oldValue` are replaced by `newValue`.

This method performs a search to find `oldValue` using the culture and case sensitivity described by `comparisonType`.

Because this method returns the modified string, you can chain together successive calls to the Replace method to perform multiple replacements on the original string. Method calls are executed from left to right. The following example provides an illustration.

```
C#
```

```
String s = "aaa";
Console.WriteLine("The initial string: '{0}'", s);
s = s.Replace("a", "b").Replace("b", "c").Replace("c", "d");
Console.WriteLine("The final string: '{0}'", s);

// The example displays the following output:
//       The initial string: 'aaa'
//       The final string: 'ddd'
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Replace(String, String, Boolean, CultureInfo)

Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string, using the provided culture and case sensitivity.

C#

```
public string Replace (string oldValue, string? newValue, bool
ignoreCase, System.Globalization.CultureInfo? culture);
```

Parameters

`oldValue` String

The string to be replaced.

`newValue` String

The string to replace all occurrences of `oldValue`.

`ignoreCase` Boolean

`true` to ignore casing when comparing; `false` otherwise.

`culture` CultureInfo

The culture to use when comparing. If `culture` is `null`, the current culture is used.

Returns

String

A string that is equivalent to the current string except that all instances of `oldValue` are replaced with `newValue`. If `oldValue` is not found in the current instance, the method returns the current instance unchanged.

Exceptions

ArgumentNullException

`oldValue` is `null`.

ArgumentException

`oldValue` is the empty string ("").

Remarks

If `newValue` is `null`, all occurrences of `oldValue` are removed.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all occurrences of `oldValue` are replaced by `newValue`.

This method performs a search to find `oldValue` using the provided `culture` and `ignoreCase` case sensitivity.

Because this method returns the modified string, you can chain together successive calls to the Replace method to perform multiple replacements on the original string. Method calls are executed from left to right. The following example provides an illustration.

C#

```
String s = "aaa";
Console.WriteLine("The initial string: '{0}'", s);
s = s.Replace("a", "b").Replace("b", "c").Replace("c", "d");
```

```
Console.WriteLine("The final string: '{0}'", s);

// The example displays the following output:
//       The initial string: 'aaa'
//       The final string: 'ddd'
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ReplaceLineEndings Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ReplaceLineEndings(String)	Replaces all newline sequences in the current string with `replacementText`.
ReplaceLineEndings()	Replaces all newline sequences in the current string with NewLine.

ReplaceLineEndings(String)

Replaces all newline sequences in the current string with `replacementText`.

C#

```
public string ReplaceLineEndings (string replacementText);
```

Parameters

`replacementText` String

The text to use as replacement.

Returns

String

A string whose contents match the current string, but with all newline sequences replaced with `replacementText`.

Remarks

This method searches for all newline sequences within the string and canonicalizes them to the newline sequence provided by `replacementText`. If `replacementText` is Empty, all newline sequences within the string will be removed.

It is not recommended that protocol parsers utilize this API. Protocol specifications often mandate specific newline sequences. For example, HTTP/1.1 (RFC 8615) mandates that the request line, status line, and headers lines end with CRLF. Since this API operates over a wide range of newline sequences, a protocol parser utilizing this API could exhibit behaviors unintended by the protocol's authors.

The list of recognized newline sequences is CR (U+000D), LF (U+000A), CRLF (U+000D U+000A), NEL (U+0085), LS (U+2028), FF (U+000C), and PS (U+2029). This list is given by the Unicode Standard, Sec. 5.8, Recommendation R4 and Table 5-2.

This method is guaranteed O(n * r) complexity, where *n* is the length of the input string, and where *r* is the length of `replacementText`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ReplaceLineEndings()

Replaces all newline sequences in the current string with NewLine.

C#

```
public string ReplaceLineEndings ();
```

Returns

String

A string whose contents match the current string, but with all newline sequences replaced with NewLine.

Remarks

This method searches for all newline sequences within the string and canonicalizes them to match the newline sequence for the current environment. For example, when running on Windows, all occurrences of non-Windows newline sequences will be replaced with the sequence CRLF. When running on Unix, all occurrences of non-Unix newline sequences will be replaced with a single LF character.

It is not recommended that protocol parsers utilize this API. Protocol specifications often mandate specific newline sequences. For example, HTTP/1.1 (RFC 8615) mandates that the request line, status line, and headers lines end with CRLF. Since this API operates over a wide range of newline sequences, a protocol parser utilizing this API could exhibit behaviors unintended by the protocol's authors.

This overload is equivalent to calling ReplaceLineEndings(String), passing NewLine as the *replacementText* parameter.

This method is guaranteed O(n) complexity, where *n* is the length of the input string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Split Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string array that contains the substrings in this instance that are delimited by elements of a specified string or Unicode character array.

Overloads

Expand table

Split(Char[])	Splits a string into substrings based on specified delimiting characters.
Split(Char, StringSplitOptions)	Splits a string into substrings based on a specified delimiting character and, optionally, options.
Split(Char[], Int32)	Splits a string into a maximum number of substrings based on specified delimiting characters.
Split(Char[], StringSplitOptions)	Splits a string into substrings based on specified delimiting characters and options.
Split(String, StringSplitOptions)	Splits a string into substrings that are based on the provided string separator.
Split(String[], StringSplitOptions)	Splits a string into substrings based on a specified delimiting string and, optionally, options.
Split(Char, Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on a specified delimiting character and, optionally, options. Splits a string into a maximum number of substrings based on the provided character separator, optionally omitting empty substrings from the result.
Split(Char[], Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on specified delimiting characters and, optionally, options.
Split(String, Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on a specified delimiting string and, optionally, options.
Split(String[], Int32, StringSplitOptions)	Splits a string into a maximum number of substrings based on specified delimiting strings and, optionally, options.

Remarks

Split is used to break a delimited string into substrings. You can use either a character array or a string array to specify zero or more delimiting characters or strings. If no delimiting characters are specified, the string is split at white-space characters.

Overloads of the Split method allow you to limit the number of substrings returned by the method (the Split(Char[], Int32) method), to specify whether to include empty strings and/or trim substrings in the result (the Split(Char[], StringSplitOptions) and Split(String[], StringSplitOptions) methods), or to do both (the Split(Char[], Int32, StringSplitOptions) and Split(String[], Int32, StringSplitOptions) methods).

> **Tip**
>
> The **Split** method is not always the best way to break a delimited string into substrings. If you don't want to extract all of the substrings of a delimited string, or if you want to parse a string based on a pattern instead of a set of delimiter characters, consider using regular expressions, or combine one of the search methods that returns the index of a character with the **Substring** method. For more information, see **Extract substrings from a string**.

Example

The following examples show three different overloads of `String.Split()`. The first example calls the Split(Char[]) overload and passes in a single delimiter.

C#

```
string s = "You win some. You lose some.";

string[] subs = s.Split(' ');

foreach (var sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: You
// Substring: win
// Substring: some.
// Substring: You
```

```
// Substring: lose
// Substring: some.
```

As you can see, the period characters (`.`) are included in two of the substrings. If you want to exclude the period characters, you can add the period character as an additional delimiting character. The next example shows how to do this.

C#

```
string s = "You win some. You lose some.";

string[] subs = s.Split(' ', '.');

foreach (var sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: You
// Substring: win
// Substring: some
// Substring:
// Substring: You
// Substring: lose
// Substring: some
// Substring:
```

The periods are gone from the substrings, but now two extra empty substrings have been included. These empty substring represent the substring between a word and the period that follows it. To omit empty substrings from the resulting array, you can call the Split(Char[], StringSplitOptions) overload and specify StringSplitOptions.RemoveEmptyEntries for the `options` parameter.

C#

```
string s = "You win some. You lose some.";
char[] separators = new char[] { ' ', '.' };

string[] subs = s.Split(separators, StringSplitOptions.RemoveEmptyEntries);

foreach (var sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
```

```
// Substring: You
// Substring: win
// Substring: some
// Substring: You
// Substring: lose
// Substring: some
```

The sections for the individual overloads of `String.Split()` contain further examples.

Split(Char[])

Splits a string into substrings based on specified delimiting characters.

```
C#

public string[] Split (params char[]? separator);
```

Parameters

`separator` Char[]

An array of delimiting characters, an empty array that contains no delimiters, or `null`.

Returns

String[]

An array whose elements contain the substrings from this instance that are delimited by one or more characters in `separator`. For more information, see the Remarks section.

Examples

The following example demonstrates how to extract individual words from a block of text by treating the space character ( ) and tab character (`\t`) as delimiters. The string being split includes both of these characters.

```
C#

string s = "Today\tI'm going to school";
string[] subs = s.Split(' ', '\t');

foreach (var sub in subs)
```

```
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: Today
// Substring: I'm
// Substring: going
// Substring: to
// Substring: school
```

Remarks

When a string is delimited by a known set of characters, you can use the Split(Char[]) method to separate it into substrings.

Delimiter characters are not included in the elements of the returned array. For example, if the separator array includes the character "-" and the value of the current string instance is "aa-bb-cc", the method returns an array that contains three elements: "aa", "bb", and "cc".

If this instance does not contain any of the characters in `separator`, the returned array consists of a single element that contains this instance.

Each element of `separator` defines a separate delimiter character. If two delimiters are adjacent, or a delimiter is found at the beginning or end of this instance, the corresponding element in the returned array contains Empty.

The following table shows some examples.

Expand table

Language	String value	Separator	Returned array
C#	"42, 12, 19"	new Char[] {',', ' '}	{"42", "", "12", "", "19"}
Visual Basic	"42, 12, 19"	Char() = {","c, " "c})	{"42", "", "12", "", "19"}
C#	"42..12..19."	new Char[] {'.'}	{"42", "", "12", "", "19", ""}
Visual Basic	"42..12..19."	Char() = {"."c}	{"42", "", "12", "", "19", ""}
C#	"Banana"	new Char[] {'.'}	{"Banana"}
Visual Basic	"Banana"	Char() = {"."c}	{"Banana"}
C#	"Darb\nSmarba"	new Char[] {}	{"Darb", "Smarba"}

Language	String value	Separator	Returned array
Visual Basic	"Darb" & vbLf & "Smarba"	Char() = {}	{"Darb", "Smarba"}
C#	"Darb\nSmarba"	null	{"Darb", "Smarba"}
Visual Basic	"Darb" & vbLf & "Smarba"	Nothing	{"Darb", "Smarba"}

The separator array

Each element of separator defines a separate delimiter that consists of a single character.

If the `separator` argument is `null` or contains no characters, the method treats white-space characters as the delimiters. White-space characters are defined by the Unicode standard, and the Char.IsWhiteSpace method returns `true` if a white-space character is passed to it.

String.Split(Char[]) and compiler overload resolution

Although the single parameter for this overload of String.Split is a character array, you can call it with a single character, as the following example shows.

C#

```
string value = "This is a short string.";
char delimiter = 's';
string[] substrings = value.Split(delimiter);
foreach (var substring in substrings)
    Console.WriteLine(substring);

// The example displays the following output:
//     Thi
//      i
//      a
//     hort
//     tring.
```

Because the `separator` parameter is decorated with the ParamArrayAttribute attribute, compilers will interpret a single character as a single-element character array. This is not the case for other String.Split overloads that include a `separator` parameter; you must explicitly pass these overloads a character array as the `separator` argument.

Comparison details

The Split(Char[]) method extracts the substrings in this string that are delimited by one or more of the characters in the `separator` array, and returns those substrings as elements of an array.

The Split(Char[]) method looks for delimiters by performing comparisons using case-sensitive ordinal sort rules. For more information about word, string, and ordinal sorts, see the System.Globalization.CompareOptions enumeration.

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method. You also have the option of using the Compare method to locate a substring within a string.

To split a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. To split a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

See also

- Char
- Concat(Object)
- Insert(Int32, String)

- Join(String, String[])
- Remove(Int32, Int32)
- Replace(Char, Char)
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Split(Char, StringSplitOptions)

Splits a string into substrings based on a specified delimiting character and, optionally, options.

C#

```
public string[] Split (char separator, StringSplitOptions options =
System.StringSplitOptions.None);
```

Parameters

`separator` Char

A character that delimits the substrings in this string.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array whose elements contain the substrings from this instance that are delimited by `separator`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Split(Char[], Int32)

Splits a string into a maximum number of substrings based on specified delimiting characters.

```
C#

public string[] Split (char[]? separator, int count);
```

Parameters

`separator` Char[]

An array of characters that delimit the substrings in this string, an empty array that contains no delimiters, or `null`.

`count` Int32

The maximum number of substrings to return.

Returns

String[]

An array whose elements contain the substrings in this instance that are delimited by one or more characters in `separator`. For more information, see the Remarks section.

Exceptions

ArgumentOutOfRangeException

`count` is negative.

Examples

The following example demonstrates how `count` can be used to limit the number of strings returned by Split.

C#

```
string name = "Alex Johnson III";

string[] subs = name.Split(null, 2);

string firstName = subs[0];
string lastName;
if (subs.Length > 1)
{
    lastName = subs[1];
}

// firstName = "Alex"
// lastName = "Johnson III"
```

Remarks

Delimiter characters are not included in the elements of the returned array.

If this instance does not contain any of the characters in `separator`, the returned array consists of a single element that contains this instance. If `count` is zero, an empty array is returned.

If the `separator` parameter is `null` or contains no characters, white-space characters are assumed to be the delimiters. White-space characters are defined by the Unicode standard and the Char.IsWhiteSpace method returns `true` if they are passed to it.

Each element of `separator` defines a separate delimiter character. If two delimiters are adjacent, or a delimiter is found at the beginning or end of this instance, the corresponding array element contains Empty.

If there are more than `count` substrings in this instance, the first `count - 1` substrings are returned in the first `count - 1` elements of the return value, and the remaining characters in this instance are returned in the last element of the return value.

If `count` is greater than the number of substrings, the available substrings are returned and no exception is thrown.

The following table shows some examples.

Expand table

Language	String value	Separator	Returned array
C#	"42, 12, 19"	new Char[] {',', ' '}	{"42", "", "12", "", "19"}
Visual Basic	"42, 12, 19"	Char() = {","c, " "c})	{"42", "", "12", "", "19"}
C#	"42..12..19."	new Char[] {'.'}	{"42", "", "12", "", "19", ""}
Visual Basic	"42..12..19."	Char() = {"."c}	{"42", "", "12", "", "19", ""}
C#	"Banana"	new Char[] {'.'}	{"Banana"}
Visual Basic	"Banana"	Char() = {"."c}	{"Banana"}
C#	"Darb\nSmarba"	new Char[] {}	{"Darb", "Smarba"}
Visual Basic	"Darb" & vbLf & "Smarba"	Char() = {}	{"Darb", "Smarba"}
C#	"Darb\nSmarba"	null	{"Darb", "Smarba"}
Visual Basic	"Darb" & vbLf & "Smarba"	Nothing	{"Darb", "Smarba"}

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method, and optionally the Compare method, to locate a substring within a string.

If you are splitting a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. If you are splitting a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

See also

- Char
- Array
- Int32
- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Remove(Int32, Int32)
- Replace(Char, Char)
- Substring(Int32)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Split(Char[], StringSplitOptions)

Splits a string into substrings based on specified delimiting characters and options.

```
C#
```

```
public string[] Split (char[]? separator, StringSplitOptions options);
```

Parameters

`separator` Char[]

An array of characters that delimit the substrings in this string, an empty array that contains no delimiters, or `null`.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array whose elements contain the substrings in this string that are delimited by one or more characters in `separator`. For more information, see the Remarks section.

Exceptions

ArgumentException

`options` is not one of the StringSplitOptions values.

Examples

The following example uses the StringSplitOptions enumeration to include or exclude substrings generated by the Split method.

C#

```
// This example demonstrates the String.Split() methods that use
// the StringSplitOptions enumeration.
string s1 = ",ONE,,TWO,,,THREE,,";
string s2 = "[stop]" +
            "ONE[stop][stop]" +
            "TWO[stop][stop][stop]" +
            "THREE[stop][stop]";
char[] charSeparators = new char[] { ',' };
string[] stringSeparators = new string[] { "[stop]" };
string[] result;
// ----------------------------------------------------------------------
--------
// Split a string delimited by characters.
```

```
// ----------------------------------------------------------------------
--------
Console.WriteLine("1) Split a string delimited by characters:\n");

// Display the original string and delimiter characters.
Console.WriteLine($"1a) The original string is \"{s1}\".");
Console.WriteLine($"The delimiter character is
'{charSeparators[0]}'.\n");

// Split a string delimited by characters and return all elements.
Console.WriteLine("1b) Split a string delimited by characters and " +
                  "return all elements:");
result = s1.Split(charSeparators, StringSplitOptions.None);
Show(result);

// Split a string delimited by characters and return all non-empty
elements.
Console.WriteLine("1c) Split a string delimited by characters and " +
                  "return all non-empty elements:");
result = s1.Split(charSeparators, StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the string and empty string before the
// delimiter and the remainder of the original string after the
delimiter.
Console.WriteLine("1d) Split a string delimited by characters and " +
                  "return 2 elements:");
result = s1.Split(charSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("1e) Split a string delimited by characters and " +
                  "return 2 non-empty elements:");
result = s1.Split(charSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// ----------------------------------------------------------------------
--------
// Split a string delimited by another string.
// ----------------------------------------------------------------------
--------
Console.WriteLine("2) Split a string delimited by another string:\n");

// Display the original string and delimiter string.
Console.WriteLine($"2a) The original string is \"{s2}\".");
Console.WriteLine($"The delimiter string is \"
{stringSeparators[0]}\".\n");

// Split a string delimited by another string and return all elements.
Console.WriteLine("2b) Split a string delimited by another string and " +
                  "return all elements:");
result = s2.Split(stringSeparators, StringSplitOptions.None);
Show(result);
```

```
// Split the original string at the delimiter and return all non-empty
elements.
Console.WriteLine("2c) Split a string delimited by another string and " +
                  "return all non-empty elements:");
result = s2.Split(stringSeparators,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the empty string before the
// delimiter and the remainder of the original string after the
delimiter.
Console.WriteLine("2d) Split a string delimited by another string and " +
                  "return 2 elements:");
result = s2.Split(stringSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("2e) Split a string delimited by another string and " +
                  "return 2 non-empty elements:");
result = s2.Split(stringSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Display the array of separated strings using a local function
void Show(string[] entries)
{
    Console.WriteLine($"The return value contains these {entries.Length}
elements:");
    foreach (string entry in entries)
    {
        Console.Write($"<{entry}>");
    }
    Console.Write("\n\n");
}

/*
This example produces the following results:

1) Split a string delimited by characters:

1a) The original string is ",ONE,,TWO,,,THREE,,".
The delimiter character is ',',

1b) Split a string delimited by characters and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

1c) Split a string delimited by characters and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

1d) Split a string delimited by characters and return 2 elements:
```

```
The return value contains these 2 elements:
<><ONE,,TWO,,,THREE,,>

1e) Split a string delimited by characters and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO,,,THREE,,>

2) Split a string delimited by another string:

2a) The original string is "[stop]ONE[stop][stop]TWO[stop][stop]
[stop]THREE[stop][stop]".
The delimiter string is "[stop]".

2b) Split a string delimited by another string and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

2c) Split a string delimited by another string and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

2d) Split a string delimited by another string and return 2 elements:
The return value contains these 2 elements:
<><ONE[stop][stop]TWO[stop][stop][stop]THREE[stop][stop]>

2e) Split a string delimited by another string and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO[stop][stop][stop]THREE[stop][stop]>

*/
```

Remarks

Delimiter characters (the characters in the `separator` array) are not included in the elements of the returned array. For example, if the `separator` array includes the character "-" and the value of the current string instance is "aa-bb-cc", the method returns an array that contains three elements: "aa", "bb", and "cc".

If this instance does not contain any of the characters in `separator`, the returned array consists of a single element that contains this instance.

If the `options` parameter is RemoveEmptyEntries and the length of this instance is zero, the method returns an empty array.

Each element of `separator` defines a separate delimiter that consists of a single character. If the `options` argument is None, and two delimiters are adjacent or a

delimiter is found at the beginning or end of this instance, the corresponding array element contains String.Empty. For example, if `separator` includes two elements, `'-'` and `'_'`, the value of the string instance is "-_aa-_", and the value of the `options` argument is None, the method returns a string array with the following five elements:

1. String.Empty, which represents the empty string that precedes the "-" character at index 0.
2. String.Empty, which represents the empty string between the "-" character at index 0 and the "_" character at index 1.
3. "aa".
4. String.Empty, which represents the empty string that follows the "-" character at index 4.
5. String.Empty, which represents the empty string that follows the "_" character at index 5.

The separator array

If the `separator` parameter is `null` or contains no characters, white-space characters are assumed to be the delimiters. White-space characters are defined by the Unicode standard and the Char.IsWhiteSpace method returns `true` if they are passed to it.

To pass `null` for the `char[] separator` parameter, you must indicate the type of the `null` to disambiguate the call from some other overloads, such as Split(String[], StringSplitOptions). The following example shows several ways to unambiguously identify this overload.

C#

```
string phrase = "The quick  brown fox";

_ = phrase.Split(default(char[]), StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split((char[]?)null, StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split(null as char[], StringSplitOptions.RemoveEmptyEntries);
```

Comparison details

The Split method extracts the substrings in this string that are delimited by one or more of the characters in the `separator` parameter, and returns those substrings as elements of an array.

The Split method looks for delimiters by performing comparisons using case-sensitive ordinal sort rules. For more information about word, string, and ordinal sorts, see the System.Globalization.CompareOptions enumeration.

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method, and optionally the Compare method, to locate a substring within a string.

If you are splitting a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. If you are splitting a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Split(String, StringSplitOptions)

Splits a string into substrings that are based on the provided string separator.

C#

```
public string[] Split (string? separator, StringSplitOptions options =
System.StringSplitOptions.None);
```

Parameters

`separator` String

A string that delimits the substrings in this string.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array whose elements contain the substrings from this instance that are delimited by `separator`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Split(String[], StringSplitOptions)

Splits a string into substrings based on a specified delimiting string and, optionally, options.

C#

```
public string[] Split (string[]? separator, StringSplitOptions options);
```

Parameters

`separator` String[]

An array of strings that delimit the substrings in this string, an empty array that contains no delimiters, or `null`.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array whose elements contain the substrings in this string that are delimited by one or more strings in `separator`. For more information, see the Remarks section.

Exceptions

ArgumentException

`options` is not one of the StringSplitOptions values.

Examples

The following example illustrates the difference in the arrays returned by calling a string's String.Split(String[], StringSplitOptions) method with its `options` parameter

equal to StringSplitOptions.None and StringSplitOptions.RemoveEmptyEntries.

C#

```
string source = "[stop]ONE[stop][stop]TWO[stop][stop][stop]THREE[stop]
[stop]";
string[] stringSeparators = new string[] { "[stop]" };
string[] result;

// Display the original string and delimiter string.
Console.WriteLine($"Splitting the string:\n   \"{source}\".");
Console.WriteLine();
Console.WriteLine($"Using the delimiter string:\n   \"
{stringSeparators[0]}\"");
Console.WriteLine();

// Split a string delimited by another string and return all elements.
result = source.Split(stringSeparators, StringSplitOptions.None);
Console.WriteLine($"Result including all elements ({result.Length}
elements):");
Console.Write("   ");
foreach (string s in result)
{
    Console.Write("'{0}' ", String.IsNullOrEmpty(s) ? "<>" : s);
}
Console.WriteLine();
Console.WriteLine();

// Split delimited by another string and return all non-empty elements.
result = source.Split(stringSeparators,
StringSplitOptions.RemoveEmptyEntries);
Console.WriteLine($"Result including non-empty elements ({result.Length}
elements):");
Console.Write("   ");
foreach (string s in result)
{
    Console.Write("'{0}' ", String.IsNullOrEmpty(s) ? "<>" : s);
}
Console.WriteLine();

// The example displays the following output:
//    Splitting the string:
//       "[stop]ONE[stop][stop]TWO[stop][stop][stop]THREE[stop][stop]".
//
//    Using the delimiter string:
//       "[stop]"
//
//    Result including all elements (9 elements):
//       '<>' 'ONE' '<>' 'TWO' '<>' '<>' 'THREE' '<>' '<>'
//
//    Result including non-empty elements (3 elements):
//       'ONE' 'TWO' 'THREE'
```

The following example defines an array of separators that include punctuation and white-space characters. Passing this array along with a value of StringSplitOptions.RemoveEmptyEntries to the Split(String[], StringSplitOptions) method returns an array that consists of the individual words from the string.

C#

```
string[] separators = { ",", ".", "!", "?", ";", ":", " " };
string value = "The handsome, energetic, young dog was playing with his
smaller, more lethargic litter mate.";
string[] words = value.Split(separators,
StringSplitOptions.RemoveEmptyEntries);
foreach (var word in words)
    Console.WriteLine(word);

// The example displays the following output:
//       The
//       handsome
//       energetic
//       young
//       dog
//       was
//       playing
//       with
//       his
//       smaller
//       more
//       lethargic
//       litter
//       mate
```

Note that the method is called with the `options` argument set to StringSplitOptions.RemoveEmptyEntries. This prevents the returned array from including String.Empty values that represent empty substring matches between punctuation marks and white-space characters.

Remarks

When a string is delimited by a known set of strings, you can use the Split method to separate it into substrings.

Delimiter strings are not included in the elements of the returned array. For example, if the `separator` array includes the string "--" and the value of the current string instance is "aa--bb--cc", the method returns an array that contains three elements: "aa", "bb", and "cc".

If this instance does not contain any of the strings in `separator`, the returned array consists of a single element that contains this instance.

If the `options` parameter is RemoveEmptyEntries and the length of this instance is zero, the method returns an empty array.

Each element of `separator` defines a separate delimiter that consists of one or more characters. If the `options` argument is None, and two delimiters are adjacent or a delimiter is found at the beginning or end of this instance, the corresponding array element contains String.Empty. For example, if `separator` includes two elements, "-" and "_", the value of the string instance is "-_aa-_", and the value of the `options` argument is None, the method returns a string array with the following five elements:

1. String.Empty, which represents the empty string that precedes the "-" substring at index 0.

2. String.Empty, which represents the empty string between the "-" substring at index 0 and the "_" substring at index 1.

3. "aa".

4. String.Empty, which represents the empty string that follows the "-" substring at index 4.

5. String.Empty, which represents the empty string that follows the "_" substring at index 5.

The separator array

If any of the elements in `separator` consists of multiple characters, the entire substring is considered a delimiter. For example, if one of the elements in `separator` is "10", attempting to split the string "This10is10a10string." returns the following four-element array: { "This", "is", "a", "string." }.

If the `separator` parameter is `null` or contains no non-empty strings, white-space characters are assumed to be the delimiters. White-space characters are defined by the Unicode standard and the Char.IsWhiteSpace method returns `true` if they are passed to it.

To pass `null` for the `string[] separator` parameter, you must indicate the type of the `null` to disambiguate the call from some other overloads, such as Split(Char[],

StringSplitOptions). The following example shows several ways to unambiguously identify this overload.

```
C#

string phrase = "The quick  brown fox";

_ = phrase.Split(default(string[]),
StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split((string[]?)null, StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split(null as string[],
StringSplitOptions.RemoveEmptyEntries);
```

Comparison details

The Split method extracts the substrings in this string that are delimited by one or more of the strings in the `separator` parameter, and returns those substrings as elements of an array.

The Split method looks for delimiters by performing comparisons using case-sensitive ordinal sort rules. For more information about word, string, and ordinal sorts, see the System.Globalization.CompareOptions enumeration.

The Split method ignores any element of `separator` whose value is `null` or the empty string ("").

To avoid ambiguous results when strings in `separator` have characters in common, the Split operation proceeds from the beginning to the end of the value of the instance, and matches the first element in `separator` that is equal to a delimiter in the instance. The order in which substrings are encountered in the instance takes precedence over the order of elements in `separator`.

For example, consider an instance whose value is "abcdef". If the first element in `separator` was "ef" and the second element was "bcde", the result of the split operation would be a string array that contains two elements, "a" and "f". This is because the substring in the instance, "bcde", is encountered and matches an element in `separator` before the substring "f" is encountered.

However, if the first element of `separator` was "bcd" and the second element was "bc", the result of the split operation would be a string array that contains two elements, "a" and "ef". This is because "bcd" is the first delimiter in `separator` that matches a delimiter in the instance. If the order of the separators was reversed so the

first element was "bc" and the second element was "bcd", the result would be a string array that contains two elements, "a" and "def".

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method, and optionally the Compare method, to locate a substring within a string.

If you are splitting a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. If you are splitting a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Split(Char, Int32, StringSplitOptions)

Splits a string into a maximum number of substrings based on a specified delimiting character and, optionally, options. Splits a string into a maximum number of substrings based on the provided character separator, optionally omitting empty substrings from the result.

C#

```
public string[] Split (char separator, int count, StringSplitOptions
options = System.StringSplitOptions.None);
```

Parameters

`separator` Char

A character that delimits the substrings in this instance.

`count` Int32

The maximum number of elements expected in the array.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array that contains at most `count` substrings from this instance that are delimited by `separator`.

Remarks

If the string has already been split `count` - 1 times, but the end of the string has not been reached, then the last string in the returned array will contain this instance's remaining trailing substring, untouched.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Split(Char[], Int32, StringSplitOptions)

Splits a string into a maximum number of substrings based on specified delimiting characters and, optionally, options.

```
C#

public string[] Split (char[]? separator, int count, StringSplitOptions
options);
```

Parameters

`separator` Char[]

An array of characters that delimit the substrings in this string, an empty array that contains no delimiters, or `null`.

`count` Int32

The maximum number of substrings to return.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array that contains the substrings in this string that are delimited by one or more characters in `separator`. For more information, see the Remarks section.

Exceptions

ArgumentOutOfRangeException

`count` is negative.

ArgumentException

`options` is not one of the StringSplitOptions values.

Examples

The following example uses the StringSplitOptions enumeration to include or exclude substrings generated by the Split method.

C#

```
// This example demonstrates the String.Split() methods that use
// the StringSplitOptions enumeration.
string s1 = ",ONE,,TWO,,,THREE,,";
string s2 = "[stop]" +
            "ONE[stop][stop]" +
            "TWO[stop][stop][stop]" +
            "THREE[stop][stop]";
char[] charSeparators = new char[] { ',' };
string[] stringSeparators = new string[] { "[stop]" };
string[] result;
// ----------------------------------------------------------------------
--------
// Split a string delimited by characters.
// ----------------------------------------------------------------------
--------
Console.WriteLine("1) Split a string delimited by characters:\n");

// Display the original string and delimiter characters.
Console.WriteLine($"1a) The original string is \"{s1}\".");
Console.WriteLine($"The delimiter character is
'{charSeparators[0]}'.\n");

// Split a string delimited by characters and return all elements.
Console.WriteLine("1b) Split a string delimited by characters and " +
                  "return all elements:");
result = s1.Split(charSeparators, StringSplitOptions.None);
Show(result);

// Split a string delimited by characters and return all non-empty
elements.
Console.WriteLine("1c) Split a string delimited by characters and " +
                  "return all non-empty elements:");
result = s1.Split(charSeparators, StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the string and empty string before the
// delimiter and the remainder of the original string after the
```

```
delimiter.
Console.WriteLine("1d) Split a string delimited by characters and " +
                  "return 2 elements:");
result = s1.Split(charSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("1e) Split a string delimited by characters and " +
                  "return 2 non-empty elements:");
result = s1.Split(charSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// ---------------------------------------------------------------------
--------
// Split a string delimited by another string.
// ---------------------------------------------------------------------
--------
Console.WriteLine("2) Split a string delimited by another string:\n");

// Display the original string and delimiter string.
Console.WriteLine($"2a) The original string is \"{s2}\".");
Console.WriteLine($"The delimiter string is \"
{stringSeparators[0]}\".\n");

// Split a string delimited by another string and return all elements.
Console.WriteLine("2b) Split a string delimited by another string and " +
                  "return all elements:");
result = s2.Split(stringSeparators, StringSplitOptions.None);
Show(result);

// Split the original string at the delimiter and return all non-empty
elements.
Console.WriteLine("2c) Split a string delimited by another string and " +
                  "return all non-empty elements:");
result = s2.Split(stringSeparators,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the empty string before the
// delimiter and the remainder of the original string after the
delimiter.
Console.WriteLine("2d) Split a string delimited by another string and " +
                  "return 2 elements:");
result = s2.Split(stringSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("2e) Split a string delimited by another string and " +
                  "return 2 non-empty elements:");
result = s2.Split(stringSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);
```

```
// Display the array of separated strings using a local function
void Show(string[] entries)
{
    Console.WriteLine($"The return value contains these {entries.Length}
elements:");
    foreach (string entry in entries)
    {
        Console.Write($"<{entry}>");
    }
    Console.Write("\n\n");
}

/*
This example produces the following results:

1) Split a string delimited by characters:

1a) The original string is ",ONE,,TWO,,,THREE,,".
The delimiter character is ',',

1b) Split a string delimited by characters and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

1c) Split a string delimited by characters and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

1d) Split a string delimited by characters and return 2 elements:
The return value contains these 2 elements:
<><ONE,,TWO,,,THREE,,>

1e) Split a string delimited by characters and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO,,,THREE,,>

2) Split a string delimited by another string:

2a) The original string is "[stop]ONE[stop][stop]TWO[stop][stop]
[stop]THREE[stop][stop]".
The delimiter string is "[stop]".

2b) Split a string delimited by another string and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

2c) Split a string delimited by another string and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

2d) Split a string delimited by another string and return 2 elements:
```

```
The return value contains these 2 elements:
<><ONE[stop][stop]TWO[stop][stop][stop]THREE[stop][stop]>

2e) Split a string delimited by another string and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO[stop][stop][stop]THREE[stop][stop]>

*/
```

Remarks

Delimiter characters are not included in the elements of the returned array.

If this instance does not contain any of the characters in `separator`, or the `count` parameter is 1, the returned array consists of a single element that contains this instance.

If the `separator` parameter is `null` or contains no characters, white-space characters are assumed to be the delimiters. White-space characters are defined by the Unicode standard and the Char.IsWhiteSpace method returns `true` if they are passed to it.

To pass `null` for the `char[] separator` parameter, you must indicate the type of the `null` to disambiguate the call from some other overloads, such as Split(String[], Int32, StringSplitOptions). The following example shows several ways to unambiguously identify this overload.

C#

```
string phrase = "The quick  brown fox";

_ = phrase.Split(default(char[]), 3,
StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split((char[]?)null, 3,
StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split(null as char[], 3,
StringSplitOptions.RemoveEmptyEntries);
```

If the `count` parameter is zero, or the `options` parameter is RemoveEmptyEntries and the length of this instance is zero, an empty array is returned.

Each element of `separator` defines a separate delimiter character. If the `options` parameter is None, and two delimiters are adjacent or a delimiter is found at the beginning or end of this instance, the corresponding array element contains Empty.

If there are more than `count` substrings in this instance, the first `count` minus 1 substrings are returned in the first `count` minus 1 elements of the return value, and the remaining characters in this instance are returned in the last element of the return value.

If `count` is greater than the number of substrings, the available substrings are returned and no exception is thrown.

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method, and optionally the Compare method, to locate a substring within a string.

If you are splitting a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. If you are splitting a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Split(String, Int32, StringSplitOptions)

Splits a string into a maximum number of substrings based on a specified delimiting string and, optionally, options.

C#

```
public string[] Split (string? separator, int count, StringSplitOptions
options = System.StringSplitOptions.None);
```

Parameters

`separator` String

A string that delimits the substrings in this instance.

`count` Int32

The maximum number of elements expected in the array.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array that contains at most `count` substrings from this instance that are delimited by `separator`.

Remarks

If the string has already been split `count` - 1 times, but the end of the string has not been reached, then the last string in the returned array will contain this instance's remaining trailing substring, untouched.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Split(String[], Int32, StringSplitOptions)

Splits a string into a maximum number of substrings based on specified delimiting strings and, optionally, options.

C#

```
public string[] Split (string[]? separator, int count, StringSplitOptions
options);
```

Parameters

`separator` String[]

The strings that delimit the substrings in this string, an empty array that contains no delimiters, or `null`.

`count` Int32

The maximum number of substrings to return.

`options` StringSplitOptions

A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Returns

String[]

An array whose elements contain the substrings in this string that are delimited by one or more strings in `separator`. For more information, see the Remarks section.

Exceptions

ArgumentOutOfRangeException

`count` is negative.

ArgumentException

`options` is not one of the StringSplitOptions values.

Examples

The following example uses the StringSplitOptions enumeration to include or exclude substrings generated by the Split method.

C#

```
// This example demonstrates the String.Split() methods that use
// the StringSplitOptions enumeration.
string s1 = ",ONE,,TWO,,,THREE,,";
string s2 = "[stop]" +
            "ONE[stop][stop]" +
            "TWO[stop][stop][stop]" +
            "THREE[stop][stop]";
char[] charSeparators = new char[] { ',' };
string[] stringSeparators = new string[] { "[stop]" };
string[] result;
// ----------------------------------------------------------------------
--------
// Split a string delimited by characters.
// ----------------------------------------------------------------------
--------
Console.WriteLine("1) Split a string delimited by characters:\n");

// Display the original string and delimiter characters.
Console.WriteLine($"1a) The original string is \"{s1}\".");
Console.WriteLine($"The delimiter character is
'{charSeparators[0]}'.\n");

// Split a string delimited by characters and return all elements.
Console.WriteLine("1b) Split a string delimited by characters and " +
                  "return all elements:");
result = s1.Split(charSeparators, StringSplitOptions.None);
Show(result);

// Split a string delimited by characters and return all non-empty
elements.
Console.WriteLine("1c) Split a string delimited by characters and " +
```

```
                  "return all non-empty elements:");
result = s1.Split(charSeparators, StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the string and empty string before the
// delimiter and the remainder of the original string after the
delimiter.
Console.WriteLine("1d) Split a string delimited by characters and " +
                  "return 2 elements:");
result = s1.Split(charSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("1e) Split a string delimited by characters and " +
                  "return 2 non-empty elements:");
result = s1.Split(charSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// ---------------------------------------------------------------------
--------
// Split a string delimited by another string.
// ---------------------------------------------------------------------
--------
Console.WriteLine("2) Split a string delimited by another string:\n");

// Display the original string and delimiter string.
Console.WriteLine($"2a) The original string is \"{s2}\".");
Console.WriteLine($"The delimiter string is \"
{stringSeparators[0]}\".\n");

// Split a string delimited by another string and return all elements.
Console.WriteLine("2b) Split a string delimited by another string and " +
                  "return all elements:");
result = s2.Split(stringSeparators, StringSplitOptions.None);
Show(result);

// Split the original string at the delimiter and return all non-empty
elements.
Console.WriteLine("2c) Split a string delimited by another string and " +
                  "return all non-empty elements:");
result = s2.Split(stringSeparators,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the empty string before the
// delimiter and the remainder of the original string after the
delimiter.
Console.WriteLine("2d) Split a string delimited by another string and " +
                  "return 2 elements:");
result = s2.Split(stringSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
```

```
// remainder of the original string after the delimiter.
Console.WriteLine("2e) Split a string delimited by another string and " +
                  "return 2 non-empty elements:");
result = s2.Split(stringSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Display the array of separated strings using a local function
void Show(string[] entries)
{
    Console.WriteLine($"The return value contains these {entries.Length}
elements:");
    foreach (string entry in entries)
    {
        Console.Write($"<{entry}>");
    }
    Console.Write("\n\n");
}

/*
This example produces the following results:

1) Split a string delimited by characters:

1a) The original string is ",ONE,,TWO,,,THREE,,".
The delimiter character is ',',

1b) Split a string delimited by characters and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

1c) Split a string delimited by characters and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

1d) Split a string delimited by characters and return 2 elements:
The return value contains these 2 elements:
<><ONE,,TWO,,,THREE,,>

1e) Split a string delimited by characters and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO,,,THREE,,>

2) Split a string delimited by another string:

2a) The original string is "[stop]ONE[stop][stop]TWO[stop][stop]
[stop]THREE[stop][stop]".
The delimiter string is "[stop]".

2b) Split a string delimited by another string and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>
```

```
2c) Split a string delimited by another string and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

2d) Split a string delimited by another string and return 2 elements:
The return value contains these 2 elements:
<><ONE[stop][stop]TWO[stop][stop][stop]THREE[stop][stop]>

2e) Split a string delimited by another string and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO[stop][stop][stop]THREE[stop][stop]>

*/
```

Remarks

Delimiter strings are not included in the elements of the returned array.

If this instance does not contain any of the strings in `separator`, or the `count` parameter is 1, the returned array consists of a single element that contains this instance.

If the `separator` parameter is `null` or contains no characters, white-space characters are assumed to be the delimiters. White-space characters are defined by the Unicode standard and the Char.IsWhiteSpace method returns `true` if they are passed to it.

To pass `null` for the `string[] separator` parameter, you must indicate the type of the `null` to disambiguate the call from some other overloads, such as Split(Char[], Int32, StringSplitOptions). The following example shows several ways to unambiguously identify this overload.

C#

```
string phrase = "The quick  brown fox";

_ = phrase.Split(default(string[]), 3,
StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split((string[]?)null, 3,
StringSplitOptions.RemoveEmptyEntries);

_ = phrase.Split(null as string[], 3,
StringSplitOptions.RemoveEmptyEntries);
```

If the `count` parameter is zero, or the `options` parameter is RemoveEmptyEntries and the length of this instance is zero, an empty array is returned.

Each element of `separator` defines a separate delimiter that consists of one or more characters. If the `options` parameter is None, and two delimiters are adjacent or a delimiter is found at the beginning or end of this instance, the corresponding array element contains Empty.

If there are more than `count` substrings in this instance, the first `count` minus 1 substrings are returned in the first `count` minus 1 elements of the return value, and the remaining characters in this instance are returned in the last element of the return value.

If `count` is greater than the number of substrings, the available substrings are returned and no exception is thrown.

The separator array

If any of the elements in `separator` consists of multiple characters, the entire substring is considered a delimiter. For example, if one of the elements in `separator` is "10", attempting to split the string "This10is10a10string." returns this four-element array: { "This", "is", "a", "string." }.

Comparison details

The Split method extracts the substrings in this string that are delimited by one or more of the strings in the `separator` parameter, and returns those substrings as elements of an array.

The Split method looks for delimiters by performing comparisons using case-sensitive ordinal sort rules. For more information about word, string, and ordinal sorts, see the System.Globalization.CompareOptions enumeration.

The Split method ignores any element of `separator` whose value is `null` or the empty string ("").

To avoid ambiguous results when strings in `separator` have characters in common, the Split method proceeds from the beginning to the end of the value of the instance, and matches the first element in `separator` that is equal to a delimiter in the instance. The order in which substrings are encountered in the instance takes precedence over the order of elements in `separator`.

For example, consider an instance whose value is "abcdef". If the first element in `separator` was "ef" and the second element was "bcde", the result of the split operation would be "a" and "f". This is because the substring in the instance, "bcde", is encountered and matches an element in `separator` before the substring "f" is encountered.

However, if the first element of `separator` was "bcd" and the second element was "bc", the result of the split operation would be "a" and "ef". This is because "bcd" is the first delimiter in `separator` that matches a delimiter in the instance. If the order of the separators was reversed so the first element was "bc" and the second element was "bcd", the result would be "a" and "def".

Performance considerations

The Split methods allocate memory for the returned array object and a String object for each array element. If your application requires optimal performance or if managing memory allocation is critical in your application, consider using the IndexOf or IndexOfAny method, and optionally the Compare method, to locate a substring within a string.

If you are splitting a string at a separator character, use the IndexOf or IndexOfAny method to locate a separator character in the string. If you are splitting a string at a separator string, use the IndexOf or IndexOfAny method to locate the first character of the separator string. Then use the Compare method to determine whether the characters after that first character are equal to the remaining characters of the separator string.

In addition, if the same set of characters is used to split strings in multiple Split method calls, consider creating a single array and referencing it in each method call. This significantly reduces the additional overhead of each method call.

Notes to Callers

In .NET Framework 3.5 and earlier versions, if the Split(Char[]) method is passed a `separator` that is `null` or contains no characters, the method uses a slightly different set of white-space characters to split the string than the Trim(Char[]) method does to trim the string. Starting with .NET Framework 4, both methods use an identical set of Unicode white-space characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.StartsWith Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

StartsWith(String, Boolean, CultureInfo)	Determines whether the beginning of this string instance matches the specified string when compared using the specified culture.
StartsWith(String, StringComparison)	Determines whether the beginning of this string instance matches the specified string when compared using the specified comparison option.
StartsWith(Char)	Determines whether this string instance starts with the specified character.
StartsWith(String)	Determines whether the beginning of this string instance matches the specified string.

StartsWith(String, Boolean, CultureInfo)

Determines whether the beginning of this string instance matches the specified string when compared using the specified culture.

C#

```
public bool StartsWith (string value, bool ignoreCase,
System.Globalization.CultureInfo? culture);
```

Parameters

`value` String

The string to compare.

`ignoreCase` Boolean

`true` to ignore case during the comparison; otherwise, `false`.

`culture` CultureInfo

Cultural information that determines how this string and `value` are compared. If `culture` is `null`, the current culture is used.

Returns

Boolean

`true` if the `value` parameter matches the beginning of this string; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example determines whether a string occurs at the beginning of another string. The StartsWith method is called several times using case sensitivity, case insensitivity, and different cultures that influence the results of the search.

C#

```
// This code example demonstrates the
// System.String.StartsWith(String, ..., CultureInfo) method.

using System;
using System.Threading;
using System.Globalization;

class Sample
{
    public static void Main()
    {
    string msg1 = "Search for the target string \"{0}\" in the string \"
{1}\".\n";
    string msg2 = "Using the {0} - \"{1}\" culture:";
    string msg3 = "  The string to search ends with the target string:
{0}";
    bool result = false;
    CultureInfo ci;

// Define a target string to search for.
// U+00c5 = LATIN CAPITAL LETTER A WITH RING ABOVE
```

```
    string capitalARing = "\u00c5";

// Define a string to search.
// The result of combining the characters LATIN SMALL LETTER A and
COMBINING
// RING ABOVE (U+0061, U+030a) is linguistically equivalent to the
character
// LATIN SMALL LETTER A WITH RING ABOVE (U+00e5).
    string aRingXYZ = "\u0061\u030a" + "xyz";

// Clear the screen and display an introduction.
    Console.Clear();

// Display the string to search for and the string to search.
    Console.WriteLine(msg1, capitalARing, aRingXYZ);

// Search using English-United States culture.
    ci = new CultureInfo("en-US");
    Console.WriteLine(msg2, ci.DisplayName, ci.Name);

    Console.WriteLine("Case sensitive:");
    result = aRingXYZ.StartsWith(capitalARing, false, ci);
    Console.WriteLine(msg3, result);

    Console.WriteLine("Case insensitive:");
    result = aRingXYZ.StartsWith(capitalARing, true, ci);
    Console.WriteLine(msg3, result);
    Console.WriteLine();

// Search using Swedish-Sweden culture.
    ci = new CultureInfo("sv-SE");
    Console.WriteLine(msg2, ci.DisplayName, ci.Name);

    Console.WriteLine("Case sensitive:");
    result = aRingXYZ.StartsWith(capitalARing, false, ci);
    Console.WriteLine(msg3, result);

    Console.WriteLine("Case insensitive:");
    result = aRingXYZ.StartsWith(capitalARing, true, ci);
    Console.WriteLine(msg3, result);
    }
}

/*
Note: This code example was executed on a console whose user interface
culture is "en-US" (English-United States).

Search for the target string "Å" in the string "a°xyz".

Using the English (United States) - "en-US" culture:
Case sensitive:
  The string to search ends with the target string: False
Case insensitive:
  The string to search ends with the target string: True
```

```
Using the Swedish (Sweden) - "sv-SE" culture:
Case sensitive:
  The string to search ends with the target string: False
Case insensitive:
  The string to search ends with the target string: False

*/
```

Remarks

This method compares the `value` parameter to the substring at the beginning of this string that is the same length as `value`, and returns a value that indicates whether they are equal. To be equal, `value` must be an empty string (String.Empty), must be a reference to this same instance, or must match the beginning of this instance.

This method performs a comparison using the specified casing and culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

StartsWith(String, StringComparison)

Determines whether the beginning of this string instance matches the specified string when compared using the specified comparison option.

C#

```
public bool StartsWith (string value, StringComparison comparisonType);
```

Parameters

`value` String

The string to compare.

`comparisonType` StringComparison

One of the enumeration values that determines how this string and `value` are compared.

Returns

Boolean

`true` if this instance begins with `value`; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`comparisonType` is not a StringComparison value.

Examples

The following example searches for the string "the" at the beginning of a longer string that begins with the word "The". As the output from the example shows, a call to the StartsWith(String, StringComparison) method that performs a culture-insensitive but case-sensitive comparison fails to match the string, while a call that performs a culture- and case-insensitive comparison matches the string.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      String title = "The House of the Seven Gables";
      String searchString = "the";
      StringComparison comparison = StringComparison.InvariantCulture;
      Console.WriteLine("'{0}':", title);
      Console.WriteLine("   Starts with '{0}' ({1:G} comparison): {2}",
                        searchString, comparison,
                        title.StartsWith(searchString, comparison));

      comparison = StringComparison.InvariantCultureIgnoreCase;
      Console.WriteLine("   Starts with '{0}' ({1:G} comparison): {2}",
                        searchString, comparison,
                        title.StartsWith(searchString, comparison));
```

```
   }
}
// The example displays the following output:
//       'The House of the Seven Gables':
//          Starts with 'the' (InvariantCulture comparison): False
//          Starts with 'the' (InvariantCultureIgnoreCase comparison):
True
```

The following example determines whether a string starts with a particular substring. It initializes a two-dimensional string array. The first element in the second dimension contains a string, and the second element contains the string to search for at the start of the first string. The results are affected by the choice of culture, whether case is ignored, and whether an ordinal comparison is performed. Note that when the string instance contains a ligature, culture-sensitive comparisons with its consecutive characters successfully match.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[,] strings = { {"ABCdef", "abc" },
                            {"ABCdef", "abc" },
                            {"œil","oe" },
                            { "læring}", "lae" } };
      for (int ctr1 = strings.GetLowerBound(0); ctr1 <=
strings.GetUpperBound(0); ctr1++)
      {
            foreach (string cmpName in
Enum.GetNames(typeof(StringComparison)))
            {
               StringComparison strCmp = (StringComparison)
Enum.Parse(typeof(StringComparison),
                                                      cmpName);
               string instance = strings[ctr1, 0];
               string value = strings[ctr1, 1];
               Console.WriteLine("{0} starts with {1}: {2} ({3}
comparison)",
                                 instance, value,
                                 instance.StartsWith(value, strCmp),
                                 strCmp);
            }
            Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       ABCdef starts with abc: False (CurrentCulture comparison)
```

```
//       ABCdef starts with abc: True (CurrentCultureIgnoreCase
comparison)
//       ABCdef starts with abc: False (InvariantCulture comparison)
//       ABCdef starts with abc: True (InvariantCultureIgnoreCase
comparison)
//       ABCdef starts with abc: False (Ordinal comparison)
//       ABCdef starts with abc: True (OrdinalIgnoreCase comparison)
//
//       ABCdef starts with abc: False (CurrentCulture comparison)
//       ABCdef starts with abc: True (CurrentCultureIgnoreCase
comparison)
//       ABCdef starts with abc: False (InvariantCulture comparison)
//       ABCdef starts with abc: True (InvariantCultureIgnoreCase
comparison)
//       ABCdef starts with abc: False (Ordinal comparison)
//       ABCdef starts with abc: True (OrdinalIgnoreCase comparison)
//
//       œil starts with oe: True (CurrentCulture comparison)
//       œil starts with oe: True (CurrentCultureIgnoreCase comparison)
//       œil starts with oe: True (InvariantCulture comparison)
//       œil starts with oe: True (InvariantCultureIgnoreCase comparison)
//       œil starts with oe: False (Ordinal comparison)
//       œil starts with oe: False (OrdinalIgnoreCase comparison)
//
//       læring} starts with lae: True (CurrentCulture comparison)
//       læring} starts with lae: True (CurrentCultureIgnoreCase
comparison)
//       læring} starts with lae: True (InvariantCulture comparison)
//       læring} starts with lae: True (InvariantCultureIgnoreCase
comparison)
//       læring} starts with lae: False (Ordinal comparison)
//       læring} starts with lae: False (OrdinalIgnoreCase comparison)
```

Remarks

The StartsWith method compares the `value` parameter to the substring at the beginning of this string and returns a value that indicates whether they are equal. To be equal, `value` must be a reference to this same string, must be the empty string ("""), or must match the beginning of this string. The type of comparison performed by the StartsWith method depends on the value of the `comparisonType` parameter. The comparison can use the conventions of the current culture (StringComparison.CurrentCulture and StringComparison.CurrentCultureIgnoreCase) or the invariant culture (StringComparison.InvariantCulture and StringComparison.InvariantCultureIgnoreCase), or it can consist of a character-by-character comparison of code points (StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase). The comparison can also be case-sensitive (StringComparison.CurrentCulture, StringComparison.InvariantCulture, or StringComparison.Ordinal), or it can ignore case

(StringComparison.CurrentCultureIgnoreCase, StringComparison.InvariantCultureIgnoreCase, StringComparison.OrdinalIgnoreCase).

See also

- StringComparison

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

StartsWith(Char)

Determines whether this string instance starts with the specified character.

C#

```
public bool StartsWith (char value);
```

Parameters

`value` Char

The character to compare.

Returns

Boolean

`true` if `value` matches the beginning of this string; otherwise, `false`.

Remarks

This method performs an ordinal (case-sensitive and culture-insensitive) comparison.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

StartsWith(String)

Determines whether the beginning of this string instance matches the specified string.

C#

```
public bool StartsWith (string value);
```

Parameters

`value` String

The string to compare.

Returns

Boolean

`true` if `value` matches the beginning of this string; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

Examples

The following example defines a `StripStartTags` method that uses the StartsWith(String) method to remove HTML start tags from the beginning of a string. Note that the `StripStartTags` method is called recursively to ensure that multiple HTML start tags at the beginning of the line are removed. The example does not remove HTML tags embedded in a string.

C#

```
using System;

public class Example
{
   public static void Main() {
      string [] strSource = { "<b>This is bold text</b>", "<H1>This is
large Text</H1>",
                "<b><i><font color=green>This has multiple tags</font>
</i></b>",
                "<b>This has <i>embedded</i> tags.</b>",
                "<This line simply begins with a lesser than symbol, it
should not be modified" };

      // Display the initial string array.
      Console.WriteLine("The original strings:");
      Console.WriteLine("---------------------");
      foreach (var s in strSource)
         Console.WriteLine(s);
      Console.WriteLine();

      Console.WriteLine("Strings after starting tags have been
stripped:");
      Console.WriteLine("----------------------------------------------
");

      // Display the strings with starting tags removed.
     foreach (var s in strSource)
        Console.WriteLine(StripStartTags(s));
   }

   private static string StripStartTags(string item)
   {
      // Determine whether a tag begins the string.
      if (item.Trim().StartsWith("<")) {
         // Find the closing tag.
         int lastLocation = item.IndexOf( ">" );
         // Remove the tag.
         if (lastLocation >= 0) {
            item =  item.Substring( lastLocation + 1 );

            // Remove any additional starting tags.
            item = StripStartTags(item);
         }
      }
```

```
        return item;
    }
}
// The example displays the following output:
//    The original strings:
//    ---------------------
//    <b>This is bold text</b>
//    <H1>This is large Text</H1>
//    <b><i><font color = green>This has multiple tags</font></i></b>
//    <b>This has <i>embedded</i> tags.</b>
//    <This line simply begins with a lesser than symbol, it should not
be modified
//
//    Strings after starting tags have been stripped:
//    -----------------------------------------------
//    This is bold text</b>
//    This is large Text</H1>
//    This has multiple tags</font></i></b>
//    This has <i>embedded</i> tags.</b>
//    <This line simply begins with a lesser than symbol, it should not
be modified
```

Remarks

This method compares `value` to the substring at the beginning of this instance that is the same length as `value`, and returns an indication whether they are equal. To be equal, `value` must be an empty string (String.Empty), must be a reference to this same instance, or must match the beginning of this instance.

This method performs a word (case-sensitive and culture-sensitive) comparison using the current culture.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string comparison methods that substitute default values and instead call methods that require parameters to be explicitly specified. To determine whether a string begins with a particular substring by using the string comparison rules of the current culture, signal your intention explicitly by calling the StartsWith(String, StringComparison) method overload with a value of CurrentCulture for its `comparisonType` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- EndsWith(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Substring Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves a substring from this instance.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overloads

Expand table

Substring(Int32)	Retrieves a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
Substring(Int32, Int32)	Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

Substring(Int32)

Retrieves a substring from this instance. The substring starts at a specified character position and continues to the end of the string.

C#

```
public string Substring (int startIndex);
```

Parameters

`startIndex` Int32

The zero-based starting character position of a substring in this instance.

Returns

String

A string that is equivalent to the substring that begins at `startIndex` in this instance, or Empty if `startIndex` is equal to the length of this instance.

Exceptions

ArgumentOutOfRangeException

`startIndex` is less than zero or greater than the length of this instance.

Examples

The following example demonstrates obtaining a substring from a string.

C#

```
string [] info = { "Name: Felica Walker", "Title: Mz.",
                   "Age: 47", "Location: Paris", "Gender: F"};
int found = 0;

Console.WriteLine("The initial values in the array are:");
foreach (string s in info)
    Console.WriteLine(s);

Console.WriteLine("\nWe want to retrieve only the key information. That
is:");
foreach (string s in info)
{
    found = s.IndexOf(": ");
    Console.WriteLine("   {0}", s.Substring(found + 2));
}

// The example displays the following output:
//       The initial values in the array are:
//       Name: Felica Walker
//       Title: Mz.
//       Age: 47
//       Location: Paris
//       Gender: F
//
//       We want to retrieve only the key information. That is:
//          Felica Walker
//          Mz.
//          47
//          Paris
//          F
```

The following example uses the Substring method to separate key/value pairs that are delimited by an equals (`=`) character.

```C#
String[] pairs = { "Color1=red", "Color2=green", "Color3=blue",
                   "Title=Code Repository" };
foreach (var pair in pairs)
{
    int position = pair.IndexOf("=");
    if (position < 0)
        continue;
    Console.WriteLine("Key: {0}, Value: '{1}'",
                      pair.Substring(0, position),
                      pair.Substring(position + 1));
}

// The example displays the following output:
//     Key: Color1, Value: 'red'
//     Key: Color2, Value: 'green'
//     Key: Color3, Value: 'blue'
//     Key: Title, Value: 'Code Repository'
```

The IndexOf method is used to get the position of the equals character in the string. The call to the Substring(Int32, Int32) method extracts the key name, which starts from the first character in the string and extends for the number of characters returned by the call to the IndexOf method. The call to the Substring(Int32) method then extracts the value assigned to the key. It starts at one character position beyond the equals character and extends to the end of the string.

Remarks

You call the Substring(Int32) method to extract a substring from a string that begins at a specified character position and ends at the end of the string. The starting character position is zero-based; in other words, the first character in the string is at index 0, not index 1. To extract a substring that begins at a specified character position and ends before the end of the string, call the Substring(Int32, Int32) method.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string that begins at the `startIndex` position in the current string.

To extract a substring that begins with a particular character or character sequence, call a method such as IndexOf or IndexOf to get the value of `startIndex`. The second

example illustrates this; it extracts a key value that begins one character position after the `=` character.

If `startIndex` is equal to zero, the method returns the original string unchanged.

See also

- Int32
- Concat(Object)
- Insert(Int32, String)
- Join(String, String[])
- Remove(Int32, Int32)
- Replace(Char, Char)
- Split(Char[])
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Substring(Int32, Int32)

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

C#

```
public string Substring (int startIndex, int length);
```

Parameters

`startIndex` Int32

The zero-based starting character position of a substring in this instance.

`length` Int32

The number of characters in the substring.

Returns

String

A string that is equivalent to the substring of length `length` that begins at `startIndex` in this instance, or Empty if `startIndex` is equal to the length of this instance and `length` is zero.

Exceptions

ArgumentOutOfRangeException

`startIndex` plus `length` indicates a position not within this instance.

-or-

`startIndex` or `length` is less than zero.

Examples

The following example illustrates a simple call to the Substring(Int32, Int32) method that extracts two characters from a string starting at the sixth character position (that is, at index five).

C#

```
String value = "This is a string.";
int startIndex = 5;
int length = 2;
String substring = value.Substring(startIndex, length);
Console.WriteLine(substring);

// The example displays the following output:
//       is
```

The following example uses the Substring(Int32, Int32) method in the following three cases to isolate substrings within a string. In two cases the substrings are used in comparisons, and in the third case an exception is thrown because invalid parameters are specified.

- It extracts the single character at the third position in the string (at index 2) and compares it with a "c". This comparison returns `true`.
- It extracts zero characters starting at the fourth position in the string (at index 3) and passes it to the IsNullOrEmpty method. This returns true because the call to the Substring method returns String.Empty.
- It attempts to extract one character starting at the fourth position in the string. Because there is no character at that position, the method call throws an ArgumentOutOfRangeException exception.

C#

```
string myString = "abc";
bool test1 = myString.Substring(2, 1).Equals("c"); // This is true.
Console.WriteLine(test1);
bool test2 = string.IsNullOrEmpty(myString.Substring(3, 0)); // This is
true.
Console.WriteLine(test2);
try
{
   string str3 = myString.Substring(3, 1); // This throws
ArgumentOutOfRangeException.
   Console.WriteLine(str3);
}
catch (ArgumentOutOfRangeException e)
{
   Console.WriteLine(e.Message);
}

// The example displays the following output:
//       True
//       True
//       Index and length must refer to a location within the string.
//       Parameter name: length
```

The following example uses the Substring method to separate key/value pairs that are delimited by an equals (`=`) character.

C#

```
String[] pairs = { "Color1=red", "Color2=green", "Color3=blue",
                   "Title=Code Repository" };
foreach (var pair in pairs)
{
    int position = pair.IndexOf("=");
    if (position < 0)
        continue;
    Console.WriteLine("Key: {0}, Value: '{1}'",
                      pair.Substring(0, position),
```

```
                    pair.Substring(position + 1));
}

// The example displays the following output:
//     Key: Color1, Value: 'red'
//     Key: Color2, Value: 'green'
//     Key: Color3, Value: 'blue'
//     Key: Title, Value: 'Code Repository'
```

The IndexOf method is used to get the position of the equals character in the string. The call to the Substring(Int32, Int32) method extracts the key name, which starts from the first character in the string and extends for the number of characters returned by the call to the IndexOf method. The call to the Substring(Int32) method then extracts the value assigned to the key. It starts at one character position beyond the equals character and extends to the end of the string.

Remarks

You call the Substring(Int32, Int32) method to extract a substring from a string that begins at a specified character position and ends before the end of the string. The starting character position is zero-based; in other words, the first character in the string is at index 0, not index 1. To extract a substring that begins at a specified character position and continues to the end of the string, call the Substring(Int32) method.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string with `length` characters starting from the `startIndex` position in the current string.

The `length` parameter represents the total number of characters to extract from the current string instance. This includes the starting character found at index `startIndex`. In other words, the Substring method attempts to extract characters from index `startIndex` to index `startIndex` + `length` - 1.

To extract a substring that begins with a particular character or character sequence, call a method such as IndexOf or LastIndexOf to get the value of `startIndex`.

If the substring should extend from `startIndex` to a specified character sequence, you can call a method such as IndexOf or LastIndexOf to get the index of the ending

character or character sequence. You can then convert that value to an index position in the string as follows:

- If you've searched for a single character that is to mark the end of the substring, the `length` parameter equals `endIndex` - `startIndex` + 1, where `endIndex` is the return value of the IndexOf or LastIndexOf method. The following example extracts a continuous block of "b" characters from a string.

  C#

  ```
  String s = "aaaaabbbcccccccdd";
  Char charRange = 'b';
  int startIndex = s.IndexOf(charRange);
  int endIndex = s.LastIndexOf(charRange);
  int length = endIndex - startIndex + 1;
  Console.WriteLine("{0}.Substring({1}, {2}) = {3}",
                  s, startIndex, length,
                  s.Substring(startIndex, length));

  // The example displays the following output:
  //       aaaaabbbcccccccdd.Substring(5, 3) = bbb
  ```

- If you've searched for multiple characters that are to mark the end of the substring, the `length` parameter equals `endIndex` + `endMatchLength` - `startIndex`, where `endIndex` is the return value of the IndexOf or LastIndexOf method, and `endMatchLength` is the length of the character sequence that marks the end of the substring. The following example extracts a block of text that contains an XML `<definition>` element.

  C#

  ```
  String s = "<term>extant<definition>still in existence</definition>
  </term>";
  String searchString = "<definition>";
  int startIndex = s.IndexOf(searchString);
  searchString = "</" + searchString.Substring(1);
  int endIndex = s.IndexOf(searchString);
  String substring = s.Substring(startIndex, endIndex +
  searchString.Length - startIndex);
  Console.WriteLine("Original string: {0}", s);
  Console.WriteLine("Substring;       {0}", substring);

  // The example displays the following output:
  //     Original string: <term>extant<definition>still in
  existence</definition></term>
  //     Substring;       <definition>still in existence</definition>
  ```

- If the character or character sequence is not included in the end of the substring, the `length` parameter equals `endIndex` - `startIndex`, where `endIndex` is the return value of the IndexOf or LastIndexOf method.

If `startIndex` is equal to zero and `length` equals the length of the current string, the method returns the original string unchanged.

See also

- Remove(Int32, Int32)
- Replace(Char, Char)
- Trim(Char[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToCharArray Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the characters in this instance to a Unicode character array.

Overloads

Expand table

ToCharArray(Int32, Int32)	Copies the characters in a specified substring in this instance to a Unicode character array.
ToCharArray()	Copies the characters in this instance to a Unicode character array.

ToCharArray(Int32, Int32)

Copies the characters in a specified substring in this instance to a Unicode character array.

C#

```
public char[] ToCharArray (int startIndex, int length);
```

Parameters

`startIndex` Int32

The starting position of a substring in this instance.

`length` Int32

The length of the substring in this instance.

Returns

Char[]

A Unicode character array whose elements are the `length` number of characters in this instance starting from character position `startIndex`.

Exceptions

ArgumentOutOfRangeException

`startIndex` or `length` is less than zero.

-or-

`startIndex` plus `length` is greater than the length of this instance.

Examples

The following example converts a substring within a string to an array of characters, then enumerates and displays the elements of the array.

C#

```
// Sample for String.ToCharArray(Int32, Int32)
using System;

class Sample {
    public static void Main() {
    string str = "012wxyz789";
    char[] arr;

    arr = str.ToCharArray(3, 4);
    Console.Write("The letters in '{0}' are: '", str);
    Console.Write(arr);
    Console.WriteLine("'");
    Console.WriteLine("Each letter in '{0}' is:", str);
    foreach (char c in arr)
        Console.WriteLine(c);
    }
}
/*
This example produces the following results:
The letters in '012wxyz789' are: 'wxyz'
Each letter in '012wxyz789' is:
w
x
y
z
*/
```

Remarks

This method copies the characters in a portion of a string to a character array. To create a string from a range of characters in a character array, call the String(Char[], Int32, Int32) constructor.

The `startIndex` parameter is zero-based. That is, the index of the first character in the string instance is zero.

If `length` is zero, the returned array is empty and has a zero length. If this instance is `null` or an empty string (""), the returned array is empty and has a zero length.

To create a byte array that contains the encoded characters in a portion of a string, instantiate the appropriate Encoding object and call its GetBytes(String, Int32, Int32, Byte[], Int32) method. Some of the standard encodings available in .NET include:

Expand table

Encoding	Object
ASCII	ASCIIEncoding
UTF-7	UTF7Encoding
UTF-8	UTF8Encoding
UTF-16	UnicodeEncoding
UTF-32	UTF32Encoding

For more information, see Character Encoding in .NET.

See also

- Array
- Char
- Int32

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToCharArray()

Copies the characters in this instance to a Unicode character array.

C#

```
public char[] ToCharArray ();
```

Returns

Char[]

A Unicode character array whose elements are the individual characters of this instance. If this instance is an empty string, the returned array is empty and has a zero length.

Examples

The following example calls the ToCharArray method to extract the characters in a string to a character array. It then displays the original string and the elements in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string s = "AaBbCcDd";
      char[] chars = s.ToCharArray();
      Console.WriteLine("Original string: {0}", s);
      Console.WriteLine("Character array:");
      for (int ctr = 0; ctr < chars.Length; ctr++)
      {
```

```
            Console.WriteLine("   {0}: {1}", ctr, chars[ctr]);
        }
    }
}

// The example displays the following output:
//     Original string: AaBbCcDd
//     Character array:
//        0: A
//        1: a
//        2: B
//        3: b
//        4: C
//        5: c
//        6: D
//        7: d
```

Remarks

This method copies each character (that is, each Char object) in a string to a character array. The first character copied is at index zero of the returned character array; the last character copied is at index Array.Length - 1.

To create a string from the characters in a character array, call the String(Char[]) constructor.

To create a byte array that contains the encoded characters in a string, instantiate the appropriate Encoding object and call its Encoding.GetBytes(String) method. Some of the standard encodings available in .NET include the following:

Expand table

Encoding	Object
ASCII	ASCIIEncoding
UTF-7	UTF7Encoding
UTF-8	UTF8Encoding
UTF-16	UnicodeEncoding
UTF-32	UTF32Encoding

For more information, see Character Encoding in .NET.

See also

- Char
- Array

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToLower Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a copy of this string converted to lowercase.

Overloads

Expand table

ToLower()	Returns a copy of this string converted to lowercase.
ToLower(CultureInfo)	Returns a copy of this string converted to lowercase, using the casing rules of the specified culture.

ToLower()

Returns a copy of this string converted to lowercase.

C#

```
public string ToLower ();
```

Returns

String

A string in lowercase.

Examples

The following example converts several mixed case strings to lowercase.

C#

```
using System;

public class ToLowerTest {
    public static void Main() {

        string [] info = {"Name", "Title", "Age", "Location", "Gender"};

        Console.WriteLine("The initial values in the array are:");
        foreach (string s in info)
            Console.WriteLine(s);

        Console.WriteLine("{0}The lowercase of these values is:",
Environment.NewLine);

        foreach (string s in info)
            Console.WriteLine(s.ToLower());

        Console.WriteLine("{0}The uppercase of these values is:",
Environment.NewLine);

        foreach (string s in info)
            Console.WriteLine(s.ToUpper());
    }
}
// The example displays the following output:
//       The initial values in the array are:
//       Name
//       Title
//       Age
//       Location
//       Gender
//
//       The lowercase of these values is:
//       name
//       title
//       age
//       location
//       gender
//
//       The uppercase of these values is:
//       NAME
//       TITLE
//       AGE
//       LOCATION
//       GENDER
```

Remarks

This method takes into account the casing rules of the current culture.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to lowercase.

The casing operation that results from calling the ToLower() method takes the casing conventions of the current culture into account. If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant methods. This produces the same result in every culture (unlike the ToLower() method) and performs more efficiently.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string casing methods that substitute default values and instead call methods that require parameters to be explicitly specified. To convert a character to lowercase by using the casing conventions of the current culture, signal your intention explicitly by calling the ToLower(CultureInfo) method overload with a value of CurrentCulture for its `culture` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- ToLowerInvariant()
- ToUpper()
- ToUpperInvariant()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToLower(CultureInfo)

Returns a copy of this string converted to lowercase, using the casing rules of the specified culture.

C#

```
public string ToLower (System.Globalization.CultureInfo? culture);
```

Parameters

`culture` CultureInfo

An object that supplies culture-specific casing rules. If `culture` is `null`, the current culture is used.

Returns

String

The lowercase equivalent of the current string.

Examples

The following example converts two strings of uppercase characters to lowercase characters using the English-United States and Turkish-Turkey cultures, then compares the lowercase strings. The uppercase strings are identical except that for each occurrence of the Unicode LATIN CAPITAL LETTER I in one string, the other string contains LATIN CAPITAL LETTER I WITH DOT ABOVE.

C#

```
// Sample for String.ToLower(CultureInfo)

using System;
using System.Globalization;

class Sample
```

```
{
    public static void Main()
    {
    String str1 = "INDIGO";
    // str2 = str1, except each 'I' is '\u0130' (Unicode LATIN CAPITAL I
WITH DOT ABOVE).
    String str2 = new String(new Char[] {'\u0130', 'N', 'D', '\u0130',
'G', 'O'});
    String str3, str4;

    Console.WriteLine();
    Console.WriteLine("str1 = '{0}'", str1);

    Console.WriteLine();
    Console.WriteLine("str1 is {0} to str2.",
         ((0 == String.CompareOrdinal(str1, str2)) ? "equal" : "not
equal"));
    CodePoints("str1", str1);
    CodePoints("str2", str2);

    Console.WriteLine();
    // str3 is a lower case copy of str2, using English-United States
culture.
    Console.WriteLine("str3 = Lower case copy of str2 using English-
United States culture.");
    str3 = str2.ToLower(new CultureInfo("en-US", false));

    // str4 is a lower case copy of str2, using Turkish-Turkey culture.
    Console.WriteLine("str4 = Lower case copy of str2 using Turkish-
Turkey culture.");
    str4 = str2.ToLower(new CultureInfo("tr-TR", false));

    // Compare the code points in str3 and str4.
    Console.WriteLine();
    Console.WriteLine("str3 is {0} to str4.",
         ((0 == String.CompareOrdinal(str3, str4)) ? "equal" : "not
equal"));
    CodePoints("str3", str3);
    CodePoints("str4", str4);
    }

    public static void CodePoints(String title, String s)
    {
    Console.Write("{0}The code points in {1} are: {0}",
Environment.NewLine, title);
    foreach (ushort u in s)
      Console.Write("{0:x4} ", u);
    Console.WriteLine();
    }
}
/*
This example produces the following results:

str1 = 'INDIGO'
```

```
str1 is not equal to str2.

The code points in str1 are:
0049 004e 0044 0049 0047 004f

The code points in str2 are:
0130 004e 0044 0130 0047 004f

str3 = Lower case copy of str2 using English-United States culture.
str4 = Lower case copy of str2 using Turkish-Turkey culture.

str3 is equal to str4.

The code points in str3 are:
0069 006e 0064 0069 0067 006f

The code points in str4 are:
0069 006e 0064 0069 0067 006f
*/
```

Remarks

The casing rules of the culture specified by the `culture` parameter determine the way the case of the string is changed.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to lowercase.

If you pass the ToLower(CultureInfo) method a CultureInfo object other than CultureInfo.InvariantCulture, the casing operation will take culture-specific rules into account. If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant method. This produces the same result in every culture and performs more efficiently.

See also

- CultureInfo
- ToUpper()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToLowerInvariant Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a copy of this String object converted to lowercase using the casing rules of the invariant culture.

C#

```
public string ToLowerInvariant ();
```

Returns

String

The lowercase equivalent of the current string.

Examples

The following example defines a string array that contains a single word in a number of languages. The ToLowerInvariant method is used to populate the elements of a parallel array with the case-insensitive version of each word. The Array.Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) method is used to sort the case-sensitive array based on the order of elements in the lowercase array to ensure that elements appear in the same order regardless of language.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] words = { "Tuesday", "Salı", "Вторник", "Mardi",
                         "Τρίτη", "Martes", "יום שלישי",
                         "الثلاثاء", "วันอังคาร" };
      // Display array in unsorted order.
      foreach (string word in words)
```

```
         Console.WriteLine(word);
      Console.WriteLine();

      // Create parallel array of words by calling ToLowerInvariant.
      string[] lowerWords = new string[words.Length];
      for (int ctr = words.GetLowerBound(0); ctr <= words.GetUpperBound(0);
ctr++)
         lowerWords[ctr] = words[ctr].ToLowerInvariant();

      // Sort the words array based on the order of lowerWords.
      Array.Sort(lowerWords, words, StringComparer.InvariantCulture);

      // Display the sorted array.
      foreach (string word in words)
         Console.WriteLine(word);
   }
}
// The example displays the following output:
//       Tuesday
//       Salı
//       Вторник
//       Mardi
//       Τρίτη
//       Martes
//       יום שלישי
//       الثلاثاء
//       วันอังคาร
//
//       Mardi
//       Martes
//       Salı
//       Tuesday
//       Τρίτη
//       Вторник
//       יום שלישי
//       الثلاثاء
//       วันอังคาร
```

Remarks

The invariant culture represents a culture that is culture-insensitive. It is associated with the English language but not with a specific country or region. For more information, see the CultureInfo.InvariantCulture property.

If your application depends on the case of a string changing in a predictable way that is unaffected by the current culture, use the ToLowerInvariant method. The ToLowerInvariant method is equivalent to `ToLower(CultureInfo.InvariantCulture)`. The method is recommended when a collection of strings must appear in a predictable order in a user interface control.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to lowercase.

If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToUpperInvariant()
- ToLower(CultureInfo)
- ToLower()
- InvariantCulture

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to a String.

Overloads

Expand table

ToString()	Returns this instance of String; no actual conversion is performed.
ToString(IFormatProvider)	Returns this instance of String; no actual conversion is performed.

ToString()

Returns this instance of String; no actual conversion is performed.

C#

```
public override string ToString ();
```

Returns

String

The current string.

Examples

The following example demonstrates the ToString method.Note that the example does not explicitly call the ToString method. Instead, the method is called implicitly by the composite formatting feature.

C#

```
using System;

class stringToString {
    public static void Main() {
    String str1 = "123";
    String str2 = "abc";

    Console.WriteLine("Original str1: {0}", str1);
    Console.WriteLine("Original str2: {0}", str2);
    Console.WriteLine("str1 same as str2?: {0}",
Object.ReferenceEquals(str1, str2));

    str2 = str1.ToString();
    Console.WriteLine();
    Console.WriteLine("New str2:      {0}", str2);
    Console.WriteLine("str1 same as str2?: {0}",
Object.ReferenceEquals(str1, str2));
    }
}
/*
This code produces the following output:
Original str1: 123
Original str2: abc
str1 same as str2?: False

New str2:      123
str1 same as str2?: True
*/
```

Remarks

Because this method simply returns the current string unchanged, there is no need to call it directly. It is usually called implicitly in a composite formatting operation, as the example shows.

See also

- Clone()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Returns this instance of String; no actual conversion is performed.

```
C#
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

(Reserved) An object that supplies culture-specific formatting information.

Returns

String

The current string.

Implements

ToString(IFormatProvider)

Remarks

`provider` is reserved, and does not currently participate in this operation.

Because this method simply returns the current string unchanged, there is no need to call it directly.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToUpper Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a copy of this string converted to uppercase.

Overloads

Expand table

ToUpper()	Returns a copy of this string converted to uppercase.
ToUpper(CultureInfo)	Returns a copy of this string converted to uppercase, using the casing rules of the specified culture.

ToUpper()

Returns a copy of this string converted to uppercase.

C#

```
public string ToUpper ();
```

Returns

String

The uppercase equivalent of the current string.

Examples

The following example calls the ToUpper method to convert a series of one-character strings that contain each character in the Basic Latin, Latin-1 Supplement, and Latin Extended-A character sets. It then displays each string whose uppercase character is different from its lowercase character.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int n = 0;
      for (int ctr = 0x20; ctr <= 0x017F; ctr++) {
         string string1 = ((char)ctr).ToString();
         string upperString = string1.ToUpper();
         if (string1 != upperString) {
            Console.Write(@"{0} (\u+{1}) --> {2} (\u+{3})         ",
                          string1,
                          Convert.ToUInt16(string1[0]).ToString("X4"),
                          upperString,

Convert.ToUInt16(upperString[0]).ToString("X4"));
            n++;
            if (n % 2 == 0) Console.WriteLine();
         }
      }
   }
}
// The example displays the following output:
//    a (\u+0061) --> A (\u+0041)         b (\u+0062) --> B (\u+0042)
//    c (\u+0063) --> C (\u+0043)         d (\u+0064) --> D (\u+0044)
//    e (\u+0065) --> E (\u+0045)         f (\u+0066) --> F (\u+0046)
//    g (\u+0067) --> G (\u+0047)         h (\u+0068) --> H (\u+0048)
//    i (\u+0069) --> I (\u+0049)         j (\u+006A) --> J (\u+004A)
//    k (\u+006B) --> K (\u+004B)         l (\u+006C) --> L (\u+004C)
//    m (\u+006D) --> M (\u+004D)         n (\u+006E) --> N (\u+004E)
//    o (\u+006F) --> O (\u+004F)         p (\u+0070) --> P (\u+0050)
//    q (\u+0071) --> Q (\u+0051)         r (\u+0072) --> R (\u+0052)
//    s (\u+0073) --> S (\u+0053)         t (\u+0074) --> T (\u+0054)
//    u (\u+0075) --> U (\u+0055)         v (\u+0076) --> V (\u+0056)
//    w (\u+0077) --> W (\u+0057)         x (\u+0078) --> X (\u+0058)
//    y (\u+0079) --> Y (\u+0059)         z (\u+007A) --> Z (\u+005A)
//    à (\u+00E0) --> À (\u+00C0)         á (\u+00E1) --> Á (\u+00C1)
//    â (\u+00E2) --> Â (\u+00C2)         ã (\u+00E3) --> Ã (\u+00C3)
//    ä (\u+00E4) --> Ä (\u+00C4)         å (\u+00E5) --> Å (\u+00C5)
//    æ (\u+00E6) --> Æ (\u+00C6)         ç (\u+00E7) --> Ç (\u+00C7)
//    è (\u+00E8) --> È (\u+00C8)         é (\u+00E9) --> É (\u+00C9)
//    ê (\u+00EA) --> Ê (\u+00CA)         ë (\u+00EB) --> Ë (\u+00CB)
//    ì (\u+00EC) --> Ì (\u+00CC)         í (\u+00ED) --> Í (\u+00CD)
//    î (\u+00EE) --> Î (\u+00CE)         ï (\u+00EF) --> Ï (\u+00CF)
//    ð (\u+00F0) --> Ð (\u+00D0)         ñ (\u+00F1) --> Ñ (\u+00D1)
//    ò (\u+00F2) --> Ò (\u+00D2)         ó (\u+00F3) --> Ó (\u+00D3)
//    ô (\u+00F4) --> Ô (\u+00D4)         õ (\u+00F5) --> Õ (\u+00D5)
//    ö (\u+00F6) --> Ö (\u+00D6)         ø (\u+00F8) --> Ø (\u+00D8)
//    ù (\u+00F9) --> Ù (\u+00D9)         ú (\u+00FA) --> Ú (\u+00DA)
//    û (\u+00FB) --> Û (\u+00DB)         ü (\u+00FC) --> Ü (\u+00DC)
//    ý (\u+00FD) --> Ý (\u+00DD)         þ (\u+00FE) --> Þ (\u+00DE)
```

```
//    ÿ (\u00FF) --> Ÿ (\u0178)          ā (\u0101) --> Ā (\u0100)
//    ă (\u0103) --> Ă (\u0102)          ą (\u0105) --> Ą (\u0104)
//    ć (\u0107) --> Ć (\u0106)          ĉ (\u0109) --> Ĉ (\u0108)
//    ċ (\u010B) --> Ċ (\u010A)          č (\u010D) --> Č (\u010C)
//    ď (\u010F) --> Ď (\u010E)          đ (\u0111) --> Đ (\u0110)
//    ē (\u0113) --> Ē (\u0112)          ĕ (\u0115) --> Ĕ (\u0114)
//    ė (\u0117) --> Ė (\u0116)          ę (\u0119) --> Ę (\u0118)
//    ě (\u011B) --> Ě (\u011A)          ĝ (\u011D) --> Ĝ (\u011C)
//    ğ (\u011F) --> Ğ (\u011E)          ġ (\u0121) --> Ġ (\u0120)
//    ģ (\u0123) --> Ģ (\u0122)          ĥ (\u0125) --> Ĥ (\u0124)
//    ħ (\u0127) --> Ħ (\u0126)          ĩ (\u0129) --> Ĩ (\u0128)
//    ī (\u012B) --> Ī (\u012A)          ĭ (\u012D) --> Ĭ (\u012C)
//     į (\u012F) --> Ī (\u012E)          ı (\u0131) --> I (\u0049)
//    ij (\u0133) --> IJ (\u0132)        ĵ (\u0135) --> Ĵ (\u0134)
//    ķ (\u0137) --> Ķ (\u0136)          ĺ (\u013A) --> Ĺ (\u0139)
//    ļ (\u013C) --> Ļ (\u013B)          ľ (\u013E) --> Ľ (\u013D)
//    l· (\u0140) --> L· (\u013F)        ł (\u0142) --> Ł (\u0141)
//    ń (\u0144) --> Ń (\u0143)          ņ (\u0146) --> Ņ (\u0145)
//    ň (\u0148) --> Ň (\u0147)          ŋ (\u014B) --> Ŋ (\u014A)
//    ō (\u014D) --> Ō (\u014C)          ŏ (\u014F) --> Ŏ (\u014E)
//    ő (\u0151) --> Ő (\u0150)          œ (\u0153) --> Œ (\u0152)
//    ŕ (\u0155) --> Ŕ (\u0154)          ŗ (\u0157) --> Ŗ (\u0156)
//    ř (\u0159) --> Ř (\u0158)          ś (\u015B) --> Ś (\u015A)
//    ŝ (\u015D) --> Ŝ (\u015C)          ş (\u015F) --> Ş (\u015E)
//    š (\u0161) --> Š (\u0160)          ţ (\u0163) --> Ţ (\u0162)
//    ť (\u0165) --> Ť (\u0164)          ŧ (\u0167) --> Ŧ (\u0166)
//    ũ (\u0169) --> Ũ (\u0168)          ū (\u016B) --> Ū (\u016A)
//    ŭ (\u016D) --> Ŭ (\u016C)          ů (\u016F) --> Ů (\u016E)
//    ű (\u0171) --> Ű (\u0170)          ŭ (\u0173) --> Ŭ (\u0172)
//    ŵ (\u0175) --> Ŵ (\u0174)          ŷ (\u0177) --> Ŷ (\u0176)
//    ź (\u017A) --> Ź (\u0179)          ż (\u017C) --> Ż (\u017B)
//    ž (\u017E) --> Ž (\u017D)
```

Remarks

This method uses the casing rules of the current culture to convert each character in the current instance to its uppercase equivalent. If a character does not have an uppercase equivalent, it is included unchanged in the returned string.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to uppercase.

The ToUpper method is often used to convert a string to uppercase so that it can be used in a case-insensitive comparison. A better method to perform case-insensitive comparison is to call a string comparison method that has a StringComparison

parameter whose value you set to StringComparison.CurrentCultureIgnoreCase for a culture-sensitive, case-insensitive comparison.

The casing operation that results from calling the ToUpper() method takes the casing conventions of the current culture into account. If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant method. This produces the same result in every culture (unlike the ToUpper() method) and performs more efficiently.

Notes to Callers

As explained in Best Practices for Using Strings, we recommend that you avoid calling string casing methods that substitute default values and instead call methods that require parameters to be explicitly specified. To convert a string to uppercase by using the casing conventions of the current culture, signal your intention explicitly by calling the ToUpper(CultureInfo) method overload with a value of CurrentCulture for its `culture` parameter. If you don't need linguistic-aware comparison, consider using Ordinal.

See also

- ToLower()
- ToLowerInvariant()
- ToUpperInvariant()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToUpper(CultureInfo)

Returns a copy of this string converted to uppercase, using the casing rules of the specified culture.

```
C#

public string ToUpper (System.Globalization.CultureInfo? culture);
```

Parameters

`culture` CultureInfo

An object that supplies culture-specific casing rules. If `culture` is `null`, the current culture is used.

Returns

String

The uppercase equivalent of the current string.

Examples

The following example converts a string of lowercase characters to two strings of uppercase characters using the English-United States and Turkish-Turkey cultures, then compares the uppercase strings. The uppercase strings are identical except that for each occurrence of the Unicode LATIN CAPITAL LETTER I in one string, the other string contains LATIN CAPITAL LETTER I WITH DOT ABOVE.

```
C#

using System;
using System.Globalization;

class Example
{
    public static void Main()
    {
       string str1 = "indigo";
       string str2, str3;

       // str2 is an uppercase copy of str1, using English-United States
culture.
       str2 = str1.ToUpper(new CultureInfo("en-US", false));
```

```
        // str3 is an uppercase copy of str1, using Turkish-Turkey
culture.
        str3 = str1.ToUpper(new CultureInfo("tr-TR", false));

        // Compare the code points and compare the uppercase strings.
        ShowCodePoints("str1", str1);
        ShowCodePoints("str2", str2);
        ShowCodePoints("str3", str3);
        Console.WriteLine("str2 is {0} to str3.",
             String.CompareOrdinal(str2, str3) == 0 ? "equal" : "not
equal");
    }

    public static void ShowCodePoints(string varName, string s)
    {
        Console.Write("{0} = {1}: ", varName, s);
        foreach (ushort u in s)
          Console.Write("{0:x4} ", u);
        Console.WriteLine();
    }
}
// This example displays the following output:
//       str1 = indigo: 0069 006e 0064 0069 0067 006f
//       str2 = INDIGO: 0049 004e 0044 0049 0047 004f
//       str3 = İNDİGO: 0130 004e 0044 0130 0047 004f
//       str2 is not equal to str3.
```

Remarks

The casing rules of the culture specified by the `culture` parameter determine the way the case of a string is changed.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to uppercase.

If you pass the ToUpper(CultureInfo) method a CultureInfo object other than CultureInfo.InvariantCulture, the casing operation will take culture-specific rules into account. If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant method. This produces the same result in every culture and performs more efficiently.

See also

- ToLower()
- ToLowerInvariant()
- ToUpperInvariant()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.ToUpperInvariant Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a copy of this String object converted to uppercase using the casing rules of the invariant culture.

C#

```
public string ToUpperInvariant ();
```

Returns

String

The uppercase equivalent of the current string.

Examples

The following example defines a string array that contains a single word in a number of languages. The ToUpperInvariant method is used to populate the elements of a parallel array with the case-insensitive version of each word. The Array.Sort<TKey,TValue>(TKey[], TValue[], IComparer<TKey>) method is used to sort the case-sensitive array based on the order of elements in the uppercase array to ensure that elements appear in the same order regardless of language.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      string[] words = { "Tuesday", "Salı", "Вторник", "Mardi",
                         "Τρίτη", "Martes", "יום שלישי",
                         "الثلاثاء", "วันอังคาร" };
      StreamWriter sw = new StreamWriter(@".\output.txt");
```

```
      // Display array in unsorted order.
      foreach (string word in words)
         sw.WriteLine(word);

      sw.WriteLine();

      // Create parallel array of words by calling ToUpperInvariant.
      string[] upperWords = new string[words.Length];
      for (int ctr = words.GetLowerBound(0); ctr <= words.GetUpperBound(0);
ctr++)
         upperWords[ctr] = words[ctr].ToUpperInvariant();

      // Sort the words array based on the order of upperWords.
      Array.Sort(upperWords, words, StringComparer.InvariantCulture);

      // Display the sorted array.
      foreach (string word in words)
         sw.WriteLine(word);

      sw.Close();
   }
}
// The example produces the following output:
//       Tuesday
//       Salı
//       Вторник
//       Mardi
//       Τρίτη
//       Martes
//       יום שלישי
//       الثلاثاء
//       วันอังคาร
//
//       Mardi
//       Martes
//       Salı
//       Tuesday
//       Τρίτη
//       Вторник
//       יום שלישי
//       الثلاثاء
//       วันอังคาร
```

Remarks

The invariant culture represents a culture that is culture-insensitive. It is associated with the English language but not with a specific country or region. For more information, see the CultureInfo.InvariantCulture property.

If your application depends on the case of a string changing in a predictable way that is unaffected by the current culture, use the ToUpperInvariant method. The ToUpperInvariant method is equivalent to `ToUpper(CultureInfo.InvariantCulture)`. The method is recommended when a collection of strings must appear in a predictable order in a user interface control.

> **Note**
>
> This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to uppercase.

If you need the lowercase or uppercase version of an operating system identifier, such as a file name, named pipe, or registry key, use the ToLowerInvariant or ToUpperInvariant methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- ToLowerInvariant()
- ToUpper()
- InvariantCulture

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

String.Trim Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new string in which all leading and trailing occurrences of a set of specified characters from the current string are removed.

Overloads

Expand table

Trim()	Removes all leading and trailing white-space characters from the current string.
Trim(Char[])	Removes all leading and trailing occurrences of a set of characters specified in an array from the current string.
Trim(Char)	Removes all leading and trailing instances of a character from the current string.

Trim()

Removes all leading and trailing white-space characters from the current string.

C#

```
public string Trim ();
```

Returns

String

The string that remains after all white-space characters are removed from the start and end of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Examples

The following example uses the String.Trim() method to remove any extra white space from strings entered by the user before concatenating them.

C#

```
using System;

public class Example
{
    public static void Main()
    {
        Console.Write("Enter your first name: ");
        string firstName = Console.ReadLine();

        Console.Write("Enter your middle name or initial: ");
        string middleName = Console.ReadLine();

        Console.Write("Enter your last name: ");
        string lastName = Console.ReadLine();

        Console.WriteLine();
        Console.WriteLine("You entered '{0}', '{1}', and '{2}'.",
                          firstName, middleName, lastName);

        string name = ((firstName.Trim() + " " +
middleName.Trim()).Trim() + " " +
                       lastName.Trim()).Trim();
        Console.WriteLine("The result is " + name + ".");

        // The following is a possible output from this example:
        //       Enter your first name:    John
        //       Enter your middle name or initial:
        //       Enter your last name:    Doe
        //
        //       You entered '   John  ', '', and '   Doe'.
        //       The result is John Doe.
    }
}
```

Remarks

The `Trim` method removes from the current string all leading and trailing white-space characters. Each leading and trailing trim operation stops when a non-white-space character is encountered. For example, if the current string is " abc xyz ", the `Trim` method returns "abc xyz". To remove white-space characters between words in a string, use .NET Regular Expressions.

ⓘ Note

If the `Trim` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading and trailing white space characters found in the current instance are removed.

If the current string equals Empty or all the characters in the current instance consist of white-space characters, the method returns Empty.

White-space characters are defined by the Unicode standard. The `Trim` method removes any leading and trailing characters that produce a return value of `true` when they are passed to the Char.IsWhiteSpace method.

Notes to Callers

The .NET Framework 3.5 SP1 and earlier versions maintain an internal list of white-space characters that this method trims. Starting with the .NET Framework 4, the method trims all Unicode white-space characters (that is, characters that produce a `true` return value when they are passed to the IsWhiteSpace(Char) method). Because of this change, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions removes two characters, ZERO WIDTH SPACE (U+200B) and ZERO WIDTH NO-BREAK SPACE (U+FEFF), that the Trim() method in the .NET Framework 4and later versions does not remove. In addition, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions does not trim three Unicode white-space characters: MONGOLIAN VOWEL SEPARATOR (U+180E), NARROW NO-BREAK SPACE (U+202F), and MEDIUM MATHEMATICAL SPACE (U+205F).

See also

- TrimEnd(Char[])
- TrimStart(Char[])
- IsWhiteSpace(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Trim(Char[])

Removes all leading and trailing occurrences of a set of characters specified in an array from the current string.

```
C#

public string Trim (params char[]? trimChars);
```

Parameters

`trimChars` Char[]

An array of Unicode characters to remove, or `null`.

Returns

String

The string that remains after all occurrences of the characters in the `trimChars` parameter are removed from the start and end of the current string. If `trimChars` is `null` or an empty array, white-space characters are removed instead. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Examples

The following example uses the `Trim(System.Char[])` method to remove space, asterisk (*), and apostrophe (') characters from a string.

```
C#

char[] charsToTrim = { '*', ' ', '\''};
string banner = "*** Much Ado About Nothing ***";
string result = banner.Trim(charsToTrim);
```

```
Console.WriteLine("Trimmed\n   {0}\nto\n   '{1}'", banner, result);

// The example displays the following output:
//       Trimmed
//          *** Much Ado About Nothing ***
//       to
//          'Much Ado About Nothing'
```

Remarks

The `Trim(System.Char[])` method removes from the current string all leading and trailing characters that are in the `trimChars` parameter. Each leading and trailing trim operation stops when a character that is not in `trimChars` is encountered. For example, if the current string is "123abc456xyz789" and `trimChars` contains the digits from "1" through "9", the `Trim(System.Char[])` method returns "abc456xyz".

> **Note**
>
> If the `Trim(System.Char[])` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading and trailing `trimChars` characters found in the current instance are removed.

If the current string equals Empty or all the characters in the current instance consist of characters in the `trimChars` array, the method returns Empty.

If `trimChars` is `null` or an empty array, this method removes any leading or trailing characters that result in the method returning `true` when they are passed to the Char.IsWhiteSpace method.

Notes to Callers

The .NET Framework 3.5 SP1 and earlier versions maintains an internal list of white-space characters that this method trims if `trimChars` is `null` or an empty array. Starting with the .NET Framework 4, if `trimChars` is `null` or an empty array, the method trims all Unicode white-space characters (that is, characters that produce a `true` return value when they are passed to the IsWhiteSpace(Char) method). Because of this change, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions removes two characters, ZERO WIDTH SPACE (U+200B) and ZERO WIDTH NO-BREAK SPACE (U+FEFF), that the Trim() method in the .NET Framework 4and later versions does not remove. In addition, the Trim() method in the .NET Framework 3.5 SP1 and

earlier versions does not trim three Unicode white-space characters: MONGOLIAN VOWEL SEPARATOR (U+180E), NARROW NO-BREAK SPACE (U+202F), and MEDIUM MATHEMATICAL SPACE (U+205F).

See also

- Char
- TrimEnd(Char[])
- TrimStart(Char[])
- IsWhiteSpace(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Trim(Char)

Removes all leading and trailing instances of a character from the current string.

C#

```
public string Trim (char trimChar);
```

Parameters

`trimChar` Char

A Unicode character to remove.

Returns

String

The string that remains after all instances of the `trimChar` character are removed from the start and end of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Remarks

The `Trim(System.Char)` method removes from the current string all leading and trailing instances of the `trimChar` character. Each leading and trailing trim operation stops when a character different from `trimChar` is encountered. For example, if `trimChar` is `-` and the current string is "---abc---xyz----", the `Trim(System.Char)` method returns "abc---xyz". To remove characters between words in a string, use .NET Regular Expressions.

> **Note**
>
> If the `Trim(System.Char)` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading and trailing `trimChar` characters found in the current instance are removed.

If the current string equals Empty or all the characters in the current instance consist of `trimChar` characters, the method returns Empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

String.TrimEnd Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TrimEnd()	Removes all the trailing white-space characters from the current string.
TrimEnd(Char)	Removes all the trailing occurrences of a character from the current string.
TrimEnd(Char[])	Removes all the trailing occurrences of a set of characters specified in an array from the current string.

TrimEnd()

Removes all the trailing white-space characters from the current string.

C#

```
public string TrimEnd ();
```

Returns

String

The string that remains after all white-space characters are removed from the end of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Remarks

The `TrimEnd` method removes from the current string all trailing white-space characters. The trim operation stops when the first non white-space character is

encountered at the end of the string. For example, if the current string is " abc xyz ", the `TrimEnd` method returns " abc xyz".

> ⓘ **Note**
>
> If the `TrimEnd` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all trailing white-space characters are removed from the current string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimEnd(Char)

Removes all the trailing occurrences of a character from the current string.

C#

```
public string TrimEnd (char trimChar);
```

Parameters

`trimChar` Char

A Unicode character to remove.

Returns

String

The string that remains after all occurrences of the `trimChar` character are removed from the end of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Remarks

The `TrimEnd(System.Char)` method removes from the current string all trailing `trimChar` characters. The trim operation stops when the first character that is not `trimChar` is encountered at the end of the string. For example, if `trimChar` is `-` and the current string is "---abc---xyz----", the `TrimEnd(System.Char)` method returns "---abc---xyz".

> **Note**
>
> If the `TrimEnd(System.Char)` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all trailing `trimChar` characters are removed from the current string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimEnd(Char[])

Removes all the trailing occurrences of a set of characters specified in an array from the current string.

C#

```
public string TrimEnd (params char[]? trimChars);
```

Parameters

`trimChars` Char[]

An array of Unicode characters to remove, or `null`.

Returns

String

The string that remains after all occurrences of the characters in the `trimChars` parameter are removed from the end of the current string. If `trimChars` is `null` or an empty array, Unicode white-space characters are removed instead. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Examples

The following example demonstrates how you can use the `TrimEnd(System.Char[])` method to trim white space or punctuation marks from the end of a string.

C#

```
string sentence = "The dog had a bone, a ball, and other toys.";
char[] charsToTrim = {',', '.', ' '};
string[] words = sentence.Split();
foreach (string word in words)
   Console.WriteLine(word.TrimEnd(charsToTrim));

// The example displays the following output:
//       The
//       dog
//       had
//       a
//       bone
//       a
//       ball
//       and
//       other
//       toys
```

Remarks

The `TrimEnd(System.Char[])` method removes from the current string all trailing characters that are in the `trimChars` parameter. The trim operation stops when the first character that is not in `trimChars` is encountered at the end of the string. For example, if the current string is "123abc456xyz789" and `trimChars` contains the digits from "1" through "9", the `TrimEnd(System.Char[])` method returns "123abc456xyz".

ⓘ Note

If the `TrimEnd(System.Char[])` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all trailing characters found in `trimChars` are removed from the current string.

Notes to Callers

The .NET Framework 3.5 SP1 and earlier versions maintains an internal list of white-space characters that this method trims if `trimChars` is `null` or an empty array. Starting with the .NET Framework 4, if `trimChars` is `null` or an empty array, the method trims all Unicode white-space characters (that is, characters that produce a `true` return value when they are passed to the IsWhiteSpace(Char) method). Because of this change, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions removes two characters, ZERO WIDTH SPACE (U+200B) and ZERO WIDTH NO-BREAK SPACE (U+FEFF), that the Trim() method in the .NET Framework 4 and later versions does not remove. In addition, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions does not trim three Unicode white-space characters: MONGOLIAN VOWEL SEPARATOR (U+180E), NARROW NO-BREAK SPACE (U+202F), and MEDIUM MATHEMATICAL SPACE (U+205F).

See also

- Char
- Trim(Char[])
- TrimStart(Char[])
- IsWhiteSpace(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.TrimStart Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TrimStart()	Removes all the leading white-space characters from the current string.
TrimStart(Char)	Removes all the leading occurrences of a specified character from the current string.
TrimStart(Char[])	Removes all the leading occurrences of a set of characters specified in an array from the current string.

TrimStart()

Removes all the leading white-space characters from the current string.

C#

```
public string TrimStart ();
```

Returns

String

The string that remains after all white-space characters are removed from the start of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Remarks

The `TrimStart` method removes from the current string all leading white-space characters. The trim operation stops when a non white-space character is

encountered. For example, if the current string is " abc xyz ", the `TrimStart` method returns "abc xyz ".

> ⓘ **Note**
>
> If the `TrimStart` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading white space characters found in the current instance are removed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimStart(Char)

Removes all the leading occurrences of a specified character from the current string.

C#

```
public string TrimStart (char trimChar);
```

Parameters

`trimChar` Char

The Unicode character to remove.

Returns

String

The string that remains after all occurrences of the `trimChar` character are removed from the start of the current string. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Remarks

The `TrimStart(System.Char)` method removes from the current string all leading `trimChar` characters. The trim operation stops when a character that is not `trimChar` is encountered. For example, if `trimChar` is `-` and the current string is "---abc---xyz-----", the `TrimStart(System.Char)` method returns "abc---xyz----".

> ⓘ **Note**
>
> If the `TrimStart(System.Char)` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading `trimChar` characters found in the current instance are removed.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TrimStart(Char[])

Removes all the leading occurrences of a set of characters specified in an array from the current string.

C#

```
public string TrimStart (params char[]? trimChars);
```

Parameters

`trimChars` Char[]

An array of Unicode characters to remove, or `null`.

Returns

String

The string that remains after all occurrences of characters in the `trimChars` parameter are removed from the start of the current string. If `trimChars` is `null` or an empty array, white-space characters are removed instead. If no characters can be trimmed from the current instance, the method returns the current instance unchanged.

Examples

The following example demonstrates the basic functionality of the TrimStart method:

C#

```
// TrimStart examples
string lineWithLeadingSpaces = "   Hello World!";
string lineWithLeadingSymbols = "$$$$Hello World!";
string lineWithLeadingUnderscores = "_____Hello World!";
string lineWithLeadingLetters = "xxxxHello World!";
string lineAfterTrimStart = string.Empty;

// Make it easy to print out and work with all of the examples
string[] lines = { lineWithLeadingSpaces, lineWithLeadingSymbols,
lineWithLeadingUnderscores, lineWithLeadingLetters };

foreach (var line in lines)
{
    Console.WriteLine($"This line has leading characters: {line}");
}
// Output:
// This line has leading characters:    Hello World!
// This line has leading characters: $$$$Hello World!
// This line has leading characters: _____Hello World!
// This line has leading characters: xxxxHello World!

// A basic demonstration of TrimStart in action
lineAfterTrimStart = lineWithLeadingSpaces.TrimStart(' ');
Console.WriteLine($"This is the result after calling TrimStart:
{lineAfterTrimStart}");
// This is the result after calling TrimStart: Hello World!

// Since TrimStart accepts a character array of leading items to be
removed as an argument,
// it's possible to do things like trim multiple pieces of data that each
have different
// leading characters,
foreach (var lineToEdit in lines)
{
    Console.WriteLine(lineToEdit.TrimStart(' ', '$', '_', 'x'));
}
// Result for each: Hello World!
```

```
// or handle pieces of data that have multiple kinds of leading
characters
var lineToBeTrimmed = "__###__ John Smith";
lineAfterTrimStart = lineToBeTrimmed.TrimStart('_', '#', ' ');
Console.WriteLine(lineAfterTrimStart);
// Result: John Smith
```

The following example uses the TrimStart method to trim white space and comment characters from lines of source code. The `StripComments` method wraps a call to TrimStart and passes it a character array that contains a space and the comment character, which is an apostrophe (') in Visual Basic and a slash (/) in C# or F#. The TrimStart method is also called to remove leading white space when evaluating whether a string is a comment.

C#

```
public static string[] StripComments(string[] lines)
{
    List<string> lineList = new List<string>();
    foreach (string line in lines)
    {
        if (line.TrimStart(' ').StartsWith("//"))
            lineList.Add(line.TrimStart(' ', '/'));
    }
    return lineList.ToArray();
}
```

The following example then illustrates a call to the `StripComments` method.

C#

```
public static void Main()
{
    string[] lines = {"using System;",
                      "",
                      "public class HelloWorld",
                      "{",
                      "   public static void Main()",
                      "   {",
                      "      // This code displays a simple greeting",
                      "      // to the console.",
                      "      Console.WriteLine(\"Hello, World.\");",
                      "   }",
                      "}"};
    Console.WriteLine("Before call to StripComments:");
    foreach (string line in lines)
        Console.WriteLine("   {0}", line);

    string[] strippedLines = StripComments(lines);
```

```
    Console.WriteLine("After call to StripComments:");
    foreach (string line in strippedLines)
        Console.WriteLine("   {0}", line);
}
// This code produces the following output to the console:
//    Before call to StripComments:
//       using System;
//
//       public class HelloWorld
//       {
//           public static void Main()
//           {
//               // This code displays a simple greeting
//               // to the console.
//               Console.WriteLine("Hello, World.");
//           }
//       }
//    After call to StripComments:
//       This code displays a simple greeting
//       to the console.
```

Remarks

The `TrimStart(System.Char[])` method removes from the current string all leading characters that are in the `trimChars` parameter. The trim operation stops when a character that is not in `trimChars` is encountered. For example, if the current string is "123abc456xyz789" and `trimChars` contains the digits from "1" through "9", the `TrimStart(System.Char[])` method returns "abc456xyz789".

> **Note**
>
> If the `TrimStart(System.Char[])` method removes any characters from the current instance, this method does not modify the value of the current instance. Instead, it returns a new string in which all leading characters that are in the `trimChars` parameter found in the current instance are removed.

Notes to Callers

The .NET Framework 3.5 SP1 and earlier versions maintains an internal list of white-space characters that this method trims if `trimChars` is `null` or an empty array. Starting with the .NET Framework 4, if `trimChars` is `null` or an empty array, the method trims all Unicode white-space characters (that is, characters that produce a `true` return value when they are passed to the IsWhiteSpace(Char) method). Because

of this change, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions removes two characters, ZERO WIDTH SPACE (U+200B) and ZERO WIDTH NO-BREAK SPACE (U+FEFF), that the Trim() method in the .NET Framework 4 and later versions does not remove. In addition, the Trim() method in the .NET Framework 3.5 SP1 and earlier versions does not trim three Unicode white-space characters: MONGOLIAN VOWEL SEPARATOR (U+180E), NARROW NO-BREAK SPACE (U+202F), and MEDIUM MATHEMATICAL SPACE (U+205F).

See also

- Char
- Trim(Char[])
- TrimEnd(Char[])
- IsWhiteSpace(Char)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.TryCopyTo(Span<Char>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the contents of this string into the destination span.

C#

```
public bool TryCopyTo (Span<char> destination);
```

Parameters

`destination` Span<Char>

The span into which to copy this string's contents.

Returns

Boolean

`true` if the data was copied; `false` if the destination was too short to fit the contents of the string.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

String.Equality(String, String) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified strings have the same value.

```C#
public static bool operator == (string? a, string? b);
```

Parameters

a String

The first string to compare, or `null`.

b String

The second string to compare, or `null`.

Returns

Boolean

`true` if the value of `a` is the same as the value of `b`; otherwise, `false`.

Examples

The following example demonstrates the equality operator.

```C#
// Example for the String Equality operator.
using System;

class EqualityOp
{
    public static void Main()
    {
        Console.WriteLine(
```

```
            "This example of the String Equality operator\n" +
            "generates the following output.\n" );

        CompareAndDisplay( "ijkl" );
        CompareAndDisplay( "ABCD" );
        CompareAndDisplay( "abcd" );
    }

    static void CompareAndDisplay( string Comparand )
    {
        String  Lower = "abcd";

        Console.WriteLine(
            "\"{0}\" == \"{1}\" ?  {2}",
            Lower, Comparand, Lower == Comparand );
    }
}

/*
This example of the String Equality operator
generates the following output.

"abcd" == "ijkl" ?  False
"abcd" == "ABCD" ?  False
"abcd" == "abcd" ?  True
*/
```

Remarks

The Equality method defines the operation of the equality operator for the String class. It enables code such as that shown in the Example section. The operator, in turn, calls the static Equals(String, String) method, which performs an ordinal (case-sensitive and culture-insensitive) comparison.

> ⓘ **Note**
>
> The Visual Basic compiler does not resolve the equality operator as a call to the **Equality** method. Instead, the equality operator wraps a call to the **Operators.CompareString** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.Implicit(String to ReadOnlySpan<Char>) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines an implicit conversion of a given string to a read-only span of characters.

C#

```
public static implicit operator ReadOnlySpan<char> (string? value);
```

Parameters

`value` String

A string to implicitly convert.

Returns

ReadOnlySpan<Char>

A new read-only span of characters representing the string.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

String.Inequality(String, String) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified strings have different values.

C#

```
public static bool operator != (string? a, string? b);
```

Parameters

a String

The first string to compare, or `null`.

b String

The second string to compare, or `null`.

Returns

Boolean

`true` if the value of `a` is different from the value of `b`; otherwise, `false`.

Examples

The following example demonstrates the inequality operator.

C#

```
// Example for the String Inequality operator.
using System;

class InequalityOp
{
    public static void Main()
```

```
    {
        Console.WriteLine(
            "This example of the String Inequality operator\n" +
            "generates the following output.\n" );

        CompareAndDisplay( "ijkl" );
        CompareAndDisplay( "ABCD" );
        CompareAndDisplay( "abcd" );
    }

    static void CompareAndDisplay( String Comparand )
    {
        String  Lower = "abcd";

        Console.WriteLine(
            "\"{0}\" != \"{1}\" ?  {2}",
            Lower, Comparand, Lower != Comparand );
    }
}

/*
This example of the String Inequality operator
generates the following output.

"abcd" != "ijkl" ?  True
"abcd" != "ABCD" ?  True
"abcd" != "abcd" ?  False
*/
```

Remarks

The Inequality method defines the operation of the inequality operator for the String class. It enables code such as that shown in the Examples section.

The Inequality operator in turn calls the static Equals(String, String) method, which performs an ordinal (case-sensitive and culture-insensitive) comparison.

> ⓘ **Note**
>
> The Visual Basic compiler does not resolve the inequality operator as a call to the **Inequality** method. Instead, the inequality operator wraps a call to the **Operators.CompareString** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IEnumerable<Char>.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator that iterates through the current String object.

C#

```
System.Collections.Generic.IEnumerator<char> IEnumerable<char>.GetEnumerator
();
```

Returns

IEnumerator<Char>

A strongly-typed enumerator that can be used to iterate through the current String object.

Implements

GetEnumerator()

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IEnumerable<T> interface object. For more information, see the GetEnumerator method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IEnumerable.GetEnumerator Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an enumerator that iterates through the current String object.

```
C#
System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An enumerator that can be used to iterate through the current string.

Implements

GetEnumerator()

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IEnumerable interface. For more information, see the IEnumerable.GetEnumerator method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current string is TrueString; `false` if the value of the current string is FalseString.

Implements

ToBoolean(IFormatProvider)

Exceptions

FormatException

The value of the current string is not TrueString or FalseString.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(String) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

C#

```
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Byte

The converted value of the current String object.

Implements

ToByte(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number greater than Byte.MaxValue or less than Byte.MinValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

```
C#
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Char

The character at index 0 in the current String object.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDateTime(IFormatProvider).

C#

```
DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

DateTime

The converted value of the current String object.

Implements

ToDateTime(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDateTime(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Decimal

The converted value of the current String object.

Implements

ToDecimal(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number less than Decimal.MinValue or than Decimal.MaxValue greater.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

C#

```
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Double

The converted value of the current String object.

Implements

ToDouble(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number less than Double.MinValue or greater than Double.MaxValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Int16

The converted value of the current String object.

Implements

ToInt16(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number greater than Int16.MaxValue or less than Int16.MinValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Int32

The converted value of the current String object.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Int64

The converted value of the current String object.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

SByte

The converted value of the current String object.

Implements

ToSByte(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number greater than SByte.MaxValue or less than SByte.MinValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSByte(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Single

The converted value of the current String object.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type of the returned object.

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

Object

The converted value of the current String object.

Implements

ToType(Type, IFormatProvider)

Exceptions

ArgumentNullException

`type` is `null`.

InvalidCastException

The value of the current String object cannot be converted to the type specified by the `type` parameter.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

UInt16

The converted value of the current String object.

Implements

ToUInt16(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number greater than UInt16.MaxValue or less than UInt16.MinValue.

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToUInt16(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

UInt32

The converted value of the current String object.

Implements

ToUInt32(IFormatProvider)

Exceptions

FormatException

The value of the current String object cannot be parsed.

OverflowException

The value of the current String object is a number greater UInt32.MaxValue or less than UInt32.MinValue

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToUInt32(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

An object that provides culture-specific formatting information.

Returns

UInt64

The converted value of the current String object.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the String instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt64(String, IFormatProvider) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

String.IParsable<String>.Parse(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a string into a value.

```
C#
static string IParsable<string>.Parse (string s, IFormatProvider provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

String

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.IParsable<String>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IParsable<string>.TryParse (string s, IFormatProvider provider,
out string result);
```

Parameters

`s` String

`provider` IFormatProvider

`result` String

Returns

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

String.ISpanParsable<String>.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a span of characters into a value.

```
C#

static string ISpanParsable<string>.Parse (ReadOnlySpan<char> s,
IFormatProvider provider);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

String

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

String.ISpanParsable<String>.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool ISpanParsable<string>.TryParse (ReadOnlySpan<char> s,
IFormatProvider provider, out string result);
```

Parameters

`s` ReadOnlySpan<Char>

`provider` IFormatProvider

`result` String

Returns

Boolean

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

StringComparer Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a string comparison operation that uses specific case and culture-based or ordinal comparison rules.

C#

```
public abstract class StringComparer :
System.Collections.Generic.IComparer<string>,
System.Collections.Generic.IEqualityComparer<string>,
System.Collections.IComparer, System.Collections.IEqualityComparer
```

Inheritance Object → StringComparer

Implements IComparer<String> , IEqualityComparer<String> , IComparer , IEqualityComparer

Examples

The following example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
```

```
    {
        // Create a list of string.
        List<string> list = new List<string>();

        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp = StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp = StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and ignores
case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");
```

```
        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
        Display(list, "Ordinal...");
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN SMALL
LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
```

```
0x131

Ordinal...
0x49
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69

*/
```

Remarks

For more information about this API, see Supplemental API remarks for StringComparer.

Constructors

Expand table

StringComparer()	Initializes a new instance of the StringComparer class.

Properties

Expand table

CurrentCulture	Gets a StringComparer object that performs a case-sensitive string comparison using the word comparison rules of the current culture.
CurrentCulture IgnoreCase	Gets a StringComparer object that performs case-insensitive string comparisons using the word comparison rules of the current culture.
InvariantCulture	Gets a StringComparer object that performs a case-sensitive string comparison using the word comparison rules of the invariant culture.
InvariantCulture IgnoreCase	Gets a StringComparer object that performs a case-insensitive string comparison using the word comparison rules of the invariant culture.

Ordinal	Gets a StringComparer object that performs a case-sensitive ordinal string comparison.
OrdinalIgnoreCase	Gets a StringComparer object that performs a case-insensitive ordinal string comparison.

Methods

Expand table

Compare(Object, Object)	When overridden in a derived class, compares two objects and returns an indication of their relative sort order.
Compare(String, String)	When overridden in a derived class, compares two strings and returns an indication of their relative sort order.
Create(CultureInfo, Boolean)	Creates a StringComparer object that compares strings according to the rules of a specified culture.
Create(CultureInfo, CompareOptions)	Creates a StringComparer object that compares strings according to the rules of a specified culture and string options.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Equals(Object, Object)	When overridden in a derived class, indicates whether two objects are equal.
Equals(String, String)	When overridden in a derived class, indicates whether two strings are equal.
FromComparison(StringComparison)	Converts the specified StringComparison instance to a StringComparer instance.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetHashCode(Object)	When overridden in a derived class, gets the hash code for the specified object.
GetHashCode(String)	When overridden in a derived class, gets the hash code for the specified string.
GetType()	Gets the Type of the current instance.

	(Inherited from Object)
IsWellKnownCultureAwareComparer(IEqualityComparer<String>, CompareInfo, CompareOptions)	Determines whether the specified IEqualityComparer<T> is a well-known culture-aware string comparer.
IsWellKnownOrdinalComparer(IEqualityComparer<String>, Boolean)	Determines whether the specified IEqualityComparer<T> is a well-known ordinal string comparer.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String
- CompareOptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the StringComparer class.

C#

```
protected StringComparer ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.CurrentCulture Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs a case-sensitive string comparison using the word comparison rules of the current culture.

C#

```
public static StringComparer CurrentCulture { get; }
```

Property Value

StringComparer

A new StringComparer object.

Examples

The following code example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
    {
        // Create a list of string.
        List<string> list = new List<string>();
```

```
        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp = StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp = StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and ignores
case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");

        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
```

```
        Display(list, "Ordinal...");
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN SMALL
LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
0x131

Ordinal...
```

```
0x49
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69

*/
```

Remarks

The StringComparer returned by the CurrentCulture property can be used when strings are linguistically relevant. For example, if strings are displayed to the user, or if strings are the result of user interaction, culture-sensitive string comparison should be used to order the string data.

The current culture is the CultureInfo object associated with the current thread.

The CurrentCulture property actually returns an instance of an anonymous class derived from the StringComparer class.

Each call to the CurrentCulture property `get` accessor returns a new StringComparer object, as the following code shows.

C#

```
private void CompareCurrentCultureStringComparer()
{
   StringComparer stringComparer1 = StringComparer.CurrentCulture;
   StringComparer stringComparer2 = StringComparer.CurrentCulture;
   // Displays false
   Console.WriteLine(StringComparer.ReferenceEquals(stringComparer1,
                                                    stringComparer2));
}
```

To improve performance, you can store the StringComparer object in a local variable rather than retrieve the value of the CurrentCulture property multiple times.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.CurrentCultureIgnoreCase Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs case-insensitive string comparisons using the word comparison rules of the current culture.

C#

```
public static StringComparer CurrentCultureIgnoreCase { get; }
```

Property Value

StringComparer

A new object for string comparison.

Remarks

The current culture is the CultureInfo object associated with the current thread; it is returned by the CultureInfo.CurrentCulture property.

The StringComparer returned by the CurrentCultureIgnoreCase property can be used when strings are linguistically relevant but their case is not. For example, if strings are displayed to the user but case is unimportant, culture-sensitive, case-insensitive string comparison should be used to order the string data.

> ⓘ Note
>
> **.NET Core running on Linux and macOS systems only:** The collation behavior for the C and Posix cultures is always case-sensitive because these cultures do not use the expected Unicode collation order. We recommend that you use a culture other than C or Posix for performing culture-sensitive, case-insensitive sorting operations.

The CurrentCultureIgnoreCase property actually returns an instance of an anonymous class derived from the StringComparer class.

Each call to the CurrentCultureIgnoreCase property `get` accessor returns a new StringComparer object, as the following code shows.

C#

```
private void CompareCurrentCultureInsensitiveStringComparer()
{
   StringComparer stringComparer1, stringComparer2;
   stringComparer1 = StringComparer.CurrentCultureIgnoreCase;
   stringComparer2 = StringComparer.CurrentCultureIgnoreCase;
   // Displays false
   Console.WriteLine(StringComparer.ReferenceEquals(stringComparer1,
                                                    stringComparer2));
}
```

To improve performance, you can store the StringComparer object in a local variable rather than retrieve the value of the CurrentCultureIgnoreCase property multiple times.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

StringComparer.InvariantCulture Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs a case-sensitive string comparison using the word comparison rules of the invariant culture.

C#

```
public static StringComparer InvariantCulture { get; }
```

Property Value

StringComparer

A new StringComparer object.

Examples

The following code example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
```

```
    {
        // Create a list of string.
        List<string> list = new List<string>();

        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp = StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp = StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and ignores
case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");
```

```
        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
        Display(list, "Ordinal...");
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN SMALL
LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
```

```
0x131

Ordinal...
0x49
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69

*/
```

Remarks

The StringComparer returned by the InvariantCulture property compares strings in a linguistically relevant manner, but it is not suitable for display in any particular culture. Its major application is to order strings in a way that will be identical across cultures.

The invariant culture is the CultureInfo object returned by the InvariantCulture property.

The InvariantCulture property actually returns an instance of an anonymous class derived from the StringComparer class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

StringComparer.InvariantCultureIgnoreCase Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs a case-insensitive string comparison using the word comparison rules of the invariant culture.

C#

```
public static StringComparer InvariantCultureIgnoreCase { get; }
```

Property Value

StringComparer

A new StringComparer object.

Remarks

The StringComparer returned by the InvariantCultureIgnoreCase property compares strings in a linguistically relevant manner that ignores case, but it is not suitable for display in any particular culture. Its major application is to order strings in a way that will be identical across cultures.

The invariant culture is the CultureInfo object returned by the InvariantCulture property.

The InvariantCultureIgnoreCase property actually returns an instance of an anonymous class derived from the StringComparer class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.Ordinal Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs a case-sensitive ordinal string comparison.

C#

```
public static StringComparer Ordinal { get; }
```

Property Value

StringComparer

A StringComparer object.

Examples

The following code example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
    {
        // Create a list of string.
        List<string> list = new List<string>();
```

```
        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp = StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp = StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and ignores
case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");

        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
        Display(list, "Ordinal...");
```

```
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN SMALL
LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
0x131

Ordinal...
0x49
```

```
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69

*/
```

Remarks

The StringComparer returned by the Ordinal property performs a simple byte comparison that is independent of language. This is most appropriate when comparing strings that are generated programmatically or when comparing case-sensitive resources such as passwords.

The Ordinal property actually returns an instance of an anonymous class derived from the StringComparer class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Best Practices for Using Strings in .NET

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

StringComparer.OrdinalIgnoreCase Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StringComparer object that performs a case-insensitive ordinal string comparison.

C#

```
public static StringComparer OrdinalIgnoreCase { get; }
```

Property Value

StringComparer

A StringComparer object.

Examples

The following code example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
```

```
    {
        // Create a list of string.
        List<string> list = new List<string>();

        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp = StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp = StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and ignores
case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");
```

```
        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
        Display(list, "Ordinal...");
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN SMALL
LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
```

```
0x131

Ordinal...
0x49
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69

*/
```

Remarks

The StringComparer returned by the OrdinalIgnoreCase property treats the characters in the strings to compare as if they were converted to uppercase using the conventions of the invariant culture, and then performs a simple byte comparison that is independent of language. This is most appropriate when comparing strings that are generated programmatically or when comparing case-insensitive resources such as paths and filenames.

The OrdinalIgnoreCase property actually returns an instance of an anonymous class derived from the StringComparer class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Best Practices for Using Strings in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.Compare Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two objects or strings and returns an indication of their relative sort order.

Overloads

Expand table

Compare(Object, Object)	When overridden in a derived class, compares two objects and returns an indication of their relative sort order.
Compare(String, String)	When overridden in a derived class, compares two strings and returns an indication of their relative sort order.

Compare(Object, Object)

When overridden in a derived class, compares two objects and returns an indication of their relative sort order.

C#

```
public int Compare (object? x, object? y);
```

Parameters

x Object

An object to compare to `y`.

y Object

An object to compare to `x`.

Returns

Int32

A signed integer that indicates the relative values of `x` and `y`, as shown in the following table.

Expand table

Value	Meaning
Less than zero	`x` precedes `y` in the sort order, or `x` is `null` and `y` is not `null`.
Zero	`x` is equal to `y`, or `x` and `y` are both `null`.
Greater than zero	`x` follows `y` in the sort order, or `y` is `null` and `x` is not `null`.

Implements

Compare(Object, Object)

Exceptions

ArgumentException

Neither `x` nor `y` is a String object, and neither `x` nor `y` implements the IComparable interface.

Remarks

The StringComparer.Compare(String, String) method is slightly more efficient than the StringComparer.Compare(Object, Object) method because no conversion of the `x` and `y` arguments is needed to perform the comparison.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Compare(String, String)

When overridden in a derived class, compares two strings and returns an indication of their relative sort order.

```
C#
public abstract int Compare (string? x, string? y);
```

Parameters

x String

A string to compare to `y`.

y String

A string to compare to `x`.

Returns

Int32

A signed integer that indicates the relative values of `x` and `y`, as shown in the following table.

Expand table

Value	Meaning
Less than zero	`x` precedes `y` in the sort order, or `x` is `null` and `y` is not `null`.
Zero	`x` is equal to `y`, or `x` and `y` are both `null`.
Greater than zero	`x` follows `y` in the sort order, or `y` is `null` and `x` is not `null`.

Implements

Compare(T, T)

Remarks

The StringComparer.Compare(String, String) method is slightly more efficient than the StringComparer.Compare(Object, Object) method because no conversion of the

`x` and `y` arguments is needed to perform the comparison.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.Create Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Create(CultureInfo, Boolean)	Creates a StringComparer object that compares strings according to the rules of a specified culture.
Create(CultureInfo, CompareOptions)	Creates a StringComparer object that compares strings according to the rules of a specified culture and string options.

Create(CultureInfo, Boolean)

Creates a StringComparer object that compares strings according to the rules of a specified culture.

C#

```
public static StringComparer Create (System.Globalization.CultureInfo
culture, bool ignoreCase);
```

Parameters

culture CultureInfo

A culture whose linguistic rules are used to perform a string comparison.

ignoreCase Boolean

`true` to specify that comparison operations be case-insensitive; `false` to specify that comparison operations be case-sensitive.

Returns

StringComparer

A new StringComparer object that performs string comparisons according to the comparison rules used by the `culture` parameter and the case rule specified by the `ignoreCase` parameter.

Exceptions

ArgumentNullException

`culture` is `null`.

Examples

The following code example demonstrates the properties and the Create method of the StringComparer class. The example illustrates how different StringComparer objects sort three versions of the Latin letter I.

C#

```
// This example demonstrates members of the
// System.StringComparer class.

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

class Sample
{
    public static void Main()
    {
        // Create a list of string.
        List<string> list = new List<string>();

        // Get the tr-TR (Turkish-Turkey) culture.
        CultureInfo turkish = new CultureInfo("tr-TR");

        // Get the culture that is associated with the current thread.
        CultureInfo thisCulture = Thread.CurrentThread.CurrentCulture;

        // Get the standard StringComparers.
        StringComparer invCmp =   StringComparer.InvariantCulture;
        StringComparer invICCmp =
StringComparer.InvariantCultureIgnoreCase;
        StringComparer currCmp = StringComparer.CurrentCulture;
        StringComparer currICCmp =
StringComparer.CurrentCultureIgnoreCase;
        StringComparer ordCmp = StringComparer.Ordinal;
```

```
        StringComparer ordICCmp = StringComparer.OrdinalIgnoreCase;

        // Create a StringComparer that uses the Turkish culture and
ignores case.
        StringComparer turkICComp = StringComparer.Create(turkish, true);

        // Define three strings consisting of different versions of the
letter I.
        // LATIN CAPITAL LETTER I (U+0049)
        string capitalLetterI = "I";

        // LATIN SMALL LETTER I (U+0069)
        string smallLetterI   = "i";

        // LATIN SMALL LETTER DOTLESS I (U+0131)
        string smallLetterDotlessI = "\u0131";

        // Add the three strings to the list.
        list.Add(capitalLetterI);
        list.Add(smallLetterI);
        list.Add(smallLetterDotlessI);

        // Display the original list order.
        Display(list, "The original order of the list entries...");

        // Sort the list using the invariant culture.
        list.Sort(invCmp);
        Display(list, "Invariant culture...");
        list.Sort(invICCmp);
        Display(list, "Invariant culture, ignore case...");

        // Sort the list using the current culture.
        Console.WriteLine("The current culture is \"{0}\".",
thisCulture.Name);
        list.Sort(currCmp);
        Display(list, "Current culture...");
        list.Sort(currICCmp);
        Display(list, "Current culture, ignore case...");

        // Sort the list using the ordinal value of the character code
points.
        list.Sort(ordCmp);
        Display(list, "Ordinal...");
        list.Sort(ordICCmp);
        Display(list, "Ordinal, ignore case...");

        // Sort the list using the Turkish culture, which treats LATIN
SMALL LETTER
        // DOTLESS I differently than LATIN SMALL LETTER I.
        list.Sort(turkICComp);
        Display(list, "Turkish culture, ignore case...");
    }

    public static void Display(List<string> lst, string title)
    {
```

```
        Char c;
        int  codePoint;
        Console.WriteLine(title);
        foreach (string s in lst)
        {
            c = s[0];
            codePoint = Convert.ToInt32(c);
            Console.WriteLine("0x{0:x}", codePoint);
        }
        Console.WriteLine();
    }
}
/*
This code example produces the following results:

The original order of the list entries...
0x49
0x69
0x131

Invariant culture...
0x69
0x49
0x131

Invariant culture, ignore case...
0x49
0x69
0x131

The current culture is "en-US".
Current culture...
0x69
0x49
0x131

Current culture, ignore case...
0x49
0x69
0x131

Ordinal...
0x49
0x69
0x131

Ordinal, ignore case...
0x69
0x49
0x131

Turkish culture, ignore case...
0x131
0x49
0x69
```

```
*/
```

See also

- CultureInfo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create(CultureInfo, CompareOptions)

Creates a StringComparer object that compares strings according to the rules of a specified culture and string options.

C#

```
public static StringComparer Create (System.Globalization.CultureInfo
culture, System.Globalization.CompareOptions options);
```

Parameters

`culture` CultureInfo

A culture whose linguistic rules are used to perform a string comparison.

`options` CompareOptions

A bitwise combination of the CompareOptions values.

Returns

StringComparer

A new StringComparer object that performs string comparisons according to the comparison rules used by the `culture` parameter and the specified `options`.

Exceptions

ArgumentNullException

`culture` is `null`.

ArgumentException

`options` has invalid flags.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, indicates whether two objects or strings are equal.

Overloads

Expand table

Equals(Object, Object)	When overridden in a derived class, indicates whether two objects are equal.
Equals(String, String)	When overridden in a derived class, indicates whether two strings are equal.

Equals(Object, Object)

When overridden in a derived class, indicates whether two objects are equal.

C#

```
public bool Equals (object? x, object? y);
```

Parameters

x Object

An object to compare to `y`.

y Object

An object to compare to `x`.

Returns

Boolean

`true` if `x` and `y` refer to the same object, or `x` and `y` are both the same type of object and those objects are equal, or both `x` and `y` are `null`; otherwise, `false`.

Implements

Equals(Object, Object)

Remarks

Because the runtime does not have to unbox `x` or `y` if they are value types or attempt to downcast `x` or `y` to strings if they are reference types, the Equals(String, String) method may be slightly more efficient than the Equals(Object, Object) method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(String, String)

When overridden in a derived class, indicates whether two strings are equal.

C#

```
public abstract bool Equals (string? x, string? y);
```

Parameters

x String

A string to compare to `y`.

y String

A string to compare to `x`.

Returns

Boolean

`true` if `x` and `y` refer to the same object, or `x` and `y` are equal, or `x` and `y` are `null`; otherwise, `false`.

Implements

Equals(T, T)

Remarks

Because the runtime does not have to unbox `x` or `y` if they are value types or attempt to downcast `x` or `y` to strings if they are reference types, the Equals(String, String) method may be slightly more efficient than the Equals(Object, Object) method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

StringComparer.FromComparison(StringComparison) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified StringComparison instance to a StringComparer instance.

C#

```
public static StringComparer FromComparison (StringComparison
comparisonType);
```

Parameters

comparisonType StringComparison

A string comparer instance to convert.

Returns

StringComparer

A StringComparer instance representing the equivalent value of the specified StringComparison instance.

Exceptions

ArgumentException

The specified string comparison is not supported.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

StringComparer.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, gets the hash code for the current StringComparer object.

Overloads

Expand table

GetHashCode(String)	When overridden in a derived class, gets the hash code for the specified string.
GetHashCode(Object)	When overridden in a derived class, gets the hash code for the specified object.

GetHashCode(String)

When overridden in a derived class, gets the hash code for the specified string.

C#

```
public abstract int GetHashCode (string obj);
```

Parameters

obj String

A string.

Returns

Int32

A 32-bit signed hash code calculated from the value of the `obj` parameter.

Implements

GetHashCode(T)

Exceptions

ArgumentException

Not enough memory is available to allocate the buffer that is required to compute the hash code.

ArgumentNullException

`obj` is `null`.

Remarks

The GetHashCode(String) method is more efficient than the GetHashCode(Object) method because the `obj` parameter does not have to be unboxed to perform the operation.

The GetHashCode(String) method allocates an amount of memory that is proportional to the size of `obj` to calculate the hash code of `obj`. In the case of large strings, trying to retrieve the hash code can throw an ArgumentException. Instead, you can use an alternate algorithm that allocates a fixed amount of memory when calculating hash codes. To use this algorithm, add the <NetFx45_CultureAwareComparerGetHashCode_LongStrings> element to the <runtime> section of your application's configuration file.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetHashCode(Object)

When overridden in a derived class, gets the hash code for the specified object.

```
C#

public int GetHashCode (object obj);
```

Parameters

`obj` Object

An object.

Returns

Int32

A 32-bit signed hash code calculated from the value of the `obj` parameter.

Implements

GetHashCode(Object)

Exceptions

ArgumentException

Not enough memory is available to allocate the buffer that is required to compute the hash code.

ArgumentNullException

`obj` is `null`.

Remarks

The GetHashCode(String) method is more efficient than the GetHashCode method because the `obj` parameter does not have to be unboxed to perform the operation.

The GetHashCode(Object) method allocates an amount of memory that is proportional to the size of `obj` to calculate the hash code of `obj`. In the case of large strings, trying to retrieve the hash code can throw an ArgumentException. Instead, you can use an alternate algorithm that allocates a fixed amount of memory when

calculating hash codes. To use this algorithm, add the <NetFx45_CultureAwareComparerGetHashCode_LongStrings> element to the <runtime> section of your application's configuration file.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.IsWellKnownCultureAwareComparer Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified IEqualityComparer<T> is a well-known culture-aware string comparer.

C#

```
public static bool IsWellKnownCultureAwareComparer
(System.Collections.Generic.IEqualityComparer<string?>? comparer, out
System.Globalization.CompareInfo? compareInfo, out
System.Globalization.CompareOptions compareOptions);
```

Parameters

comparer IEqualityComparer<String>

The comparer to query.

compareInfo CompareInfo

When this method returns, contains a value indicating which CompareInfo was used to create `comparer`. Set to `null` if this method returns `false`.

compareOptions CompareOptions

When this method returns, contains a value indicating which CompareOptions was used to create `comparer`. Set to None if this method returns `false`.

Returns

Boolean

`true` if `comparer` is a well-known culture-aware string comparer; otherwise, `false`.

Remarks

A "well-known culture-aware comparer" describes a comparer that's tied to a specific CompareInfo using some defined CompareOptions. To create a StringComparer instance wrapped around a CompareInfo and CompareOptions, use GetStringComparer(CompareInfo, CompareOptions). This method returns `false` when given Ordinal and other non-linguistic comparers as input.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparer.IsWellKnownOrdinalComparer Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified IEqualityComparer<T> is a well-known ordinal string comparer.

C#

```
public static bool IsWellKnownOrdinalComparer
(System.Collections.Generic.IEqualityComparer<string?>? comparer, out bool
ignoreCase);
```

Parameters

comparer IEqualityComparer<String>

The comparer to query.

ignoreCase Boolean

When this method returns, contains a value stating whether `comparer` is case-insensitive. Set to `false` if this method returns `false`.

Returns

Boolean

`true` if `comparer` is a well-known ordinal string comparer; otherwise, `false`.

Remarks

A "well-known ordinal comparer" describes a comparer that behaves identically to Ordinal when passed to Dictionary<TKey,TValue> or HashSet<T>. For example, Default is a well-known ordinal comparer because a Dictionary<TKey,TValue> given Default as a constructor argument will behave identically to a Dictionary<TKey,TValue> given

Ordinal as a constructor argument. If `ignoreCase` is `true` on method exit, then `comparer` behaves identically to OrdinalIgnoreCase when passed to the constructor of such a collection.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringComparison Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the culture, case, and sort rules to be used by certain overloads of the Compare(String, String) and Equals(Object) methods.

```
C#
public enum StringComparison
```

Inheritance Object → ValueType → Enum → StringComparison

Fields

Expand table

CurrentCulture	0	Compare strings using culture-sensitive sort rules and the current culture.
CurrentCultureIgnoreCase	1	Compare strings using culture-sensitive sort rules, the current culture, and ignoring the case of the strings being compared.
InvariantCulture	2	Compare strings using culture-sensitive sort rules and the invariant culture.
InvariantCultureIgnoreCase	3	Compare strings using culture-sensitive sort rules, the invariant culture, and ignoring the case of the strings being compared.
Ordinal	4	Compare strings using ordinal (binary) sort rules.
OrdinalIgnoreCase	5	Compare strings using ordinal (binary) sort rules and ignoring the case of the strings being compared.

Examples

The following example compares three sets of strings by using each member of the StringComparison enumeration. The comparisons use the conventions of the English

(United States), Thai (Thailand), and Turkish (Turkey) cultures. Note that the strings "a" and "a-" are considered equivalent in the "th-TH" culture but not in the others, while "i" and "İ" are considered equivalent in the "tr-TR" culture when case is ignored but not in the other cultures.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example3
{
    public static void Main()
    {
        String[] cultureNames = { "en-US", "th-TH", "tr-TR" };
        String[] strings1 = { "a", "i", "case", };
        String[] strings2 = { "a-", "\u0130", "Case" };
        StringComparison[] comparisons =
(StringComparison[])Enum.GetValues(typeof(StringComparison));

        foreach (var cultureName in cultureNames)
        {
            Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture(cultureName);
            Console.WriteLine("Current Culture: {0}",
CultureInfo.CurrentCulture.Name);
            for (int ctr = 0; ctr <= strings1.GetUpperBound(0); ctr++)
            {
                foreach (var comparison in comparisons)
                    Console.WriteLine("   {0} = {1} ({2}): {3}",
strings1[ctr],
                                      strings2[ctr], comparison,
                                      String.Equals(strings1[ctr],
strings2[ctr], comparison));

                Console.WriteLine();
            }
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//    Current Culture: en-US
//       a = a- (CurrentCulture): False
//       a = a- (CurrentCultureIgnoreCase): False
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): False
```

```
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
//
//
//    Current Culture: th-TH
//       a = a- (CurrentCulture): True
//       a = a- (CurrentCultureIgnoreCase): True
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): False
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
//
//
//    Current Culture: tr-TR
//       a = a- (CurrentCulture): False
//       a = a- (CurrentCultureIgnoreCase): False
//       a = a- (InvariantCulture): False
//       a = a- (InvariantCultureIgnoreCase): False
//       a = a- (Ordinal): False
//       a = a- (OrdinalIgnoreCase): False
//
//       i = İ (CurrentCulture): False
//       i = İ (CurrentCultureIgnoreCase): True
//       i = İ (InvariantCulture): False
//       i = İ (InvariantCultureIgnoreCase): False
//       i = İ (Ordinal): False
//       i = İ (OrdinalIgnoreCase): False
//
//       case = Case (CurrentCulture): False
//       case = Case (CurrentCultureIgnoreCase): True
//       case = Case (InvariantCulture): False
//       case = Case (InvariantCultureIgnoreCase): True
```

```
//       case = Case (Ordinal): False
//       case = Case (OrdinalIgnoreCase): True
```

Remarks

The StringComparison enumeration is used to specify whether a string comparison should use the current culture or the invariant culture, word or ordinal sort rules, and be case-sensitive or case-insensitive.

> ⓘ **Important**
>
> When you call a string comparison method such as **String.Compare**, **String.Equals**, or **String.IndexOf**, you should always call an overload that includes a parameter of type **StringComparison** so that you can specify the type of comparison that the method performs. For more information, see **Best Practices for Using Strings**.

An operation that uses word sort rules performs a culture-sensitive comparison wherein certain nonalphanumeric Unicode characters might have special weights assigned to them. Using word sort rules and the conventions of a specific culture, the hyphen ("-") might have a very small weight assigned to it so that "coop" and "co-op" appear next to each other in a sorted list.

> ⓘ **Note**
>
> **.NET Core running on Linux and macOS systems only:** The collation behavior for the C and Posix cultures is always case-sensitive because these cultures do not use the expected Unicode collation order. We recommend that you use a culture other than C or Posix for performing culture-sensitive, case-insensitive sorting operations.

An operation that uses ordinal sort rules performs a comparison based on the numeric value (Unicode code point) of each Char in the string. An ordinal comparison is fast but culture-insensitive. When you use ordinal sort rules to sort strings that start with Unicode characters (U+), the string U+xxxx comes before the string U+yyyy if the value of xxxx is numerically less than yyyy.

For more information about comparisons, see the System.String class remarks. For more information about culture, see the System.Globalization.CultureInfo class remarks. For guidelines on when to use ordinal or culture-sensitive comparison rules or the rules of the invariant culture, see Best Practices for Using Strings. For a set of text files that contain information on the character weights used in sorting and comparison

operations for Windows operating systems, see Sorting Weight Tables. For the sort weight table for Linux and macOS, see the Default Unicode Collation Element Table.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String
- Compare(String, String)
- Compare(String, String)
- CompareOrdinal(String, String)
- InvariantCulture

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringNormalizationExtensions Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides extension methods to work with string normalization.

C#

```
public static class StringNormalizationExtensions
```

Inheritance Object → StringNormalizationExtensions

Remarks

The extension methods of the StringNormalizationExtensions class are designed to work with .NET implementations that do not support the String.IsNormalized and String.Normalize methods. However, these methods are included in the String class in .NET Standard 2.0 and are therefore available on all .NET implementations that support the .NET Standard 2.0. For more information, see .NET Standard.

Methods

Expand table

IsNormalized(String)	Indicates whether the specified string is in Unicode normalization form C.
IsNormalized(String, NormalizationForm)	Indicates whether a string is in a specified Unicode normalization form.
Normalize(String)	Normalizes a string to a Unicode normalization form C.
Normalize(String, NormalizationForm)	Normalizes a string to the specified Unicode normalization form.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringNormalizationExtensions.IsNormalized Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a string is normalized.

Overloads

Expand table

IsNormalized(String)	Indicates whether the specified string is in Unicode normalization form C.
IsNormalized(String, NormalizationForm)	Indicates whether a string is in a specified Unicode normalization form.

IsNormalized(String)

Indicates whether the specified string is in Unicode normalization form C.

Parameters

String

A string.

Returns

`true` if `value` is in normalization form C; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`value` contains invalid Unicode characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

IsNormalized(String, NormalizationForm)

Indicates whether a string is in a specified Unicode normalization form.

Parameters

String

A string.

`normalizationForm` NormalizationForm

A Unicode normalization form.

Returns

`true` if `value` is in normalization form `normalizationForm`; otherwise, `false`.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`value` contains invalid Unicode characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringNormalizationExtensions.Normalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Normalizes a string.

Overloads

Expand table

Normalize(String)	Normalizes a string to a Unicode normalization form C.
Normalize(String, NormalizationForm)	Normalizes a string to the specified Unicode normalization form.

Normalize(String)

Normalizes a string to a Unicode normalization form C.

Parameters

String

The string to normalize.

Returns

A new string whose textual value is the same as `value` but whose binary representation is in Unicode normalization form C.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`value` contains invalid Unicode characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Normalize(String, NormalizationForm)

Normalizes a string to the specified Unicode normalization form.

Parameters

String

The string to normalize.

`normalizationForm` NormalizationForm

The Unicode normalization form.

Returns

A new string whose textual value is the same as `value` but whose binary representation is in the `normalizationForm` normalization form.

Exceptions

ArgumentNullException

`value` is `null`.

ArgumentException

`value` contains invalid Unicode characters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

StringSplitOptions Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies options for applicable Split method overloads, such as whether to omit empty substrings from the returned array or trim whitespace from substrings.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum StringSplitOptions
```

Inheritance Object → ValueType → Enum → StringSplitOptions

Attributes FlagsAttribute

Fields

Expand table

None	0	Use the default options when splitting strings.
RemoveEmptyEntries	1	Omit array elements that contain an empty string from the result. If RemoveEmptyEntries and TrimEntries are specified together, then substrings that consist only of white-space characters are also removed from the result.
TrimEntries	2	Trim white-space characters from each substring in the result. This field is available in .NET 5 and later versions only. If RemoveEmptyEntries and TrimEntries are specified together, then substrings that consist only of white-space characters are also removed from the result.

Examples

The following example shows how the `StringSplitOptions` enumeration is used to include or exclude substrings generated by the String.Split method:

C#

```
// This example demonstrates the String.Split() methods that use
// the StringSplitOptions enumeration.
string s1 = ",ONE,,TWO,,,THREE,,";
string s2 = "[stop]" +
            "ONE[stop][stop]" +
            "TWO[stop][stop][stop]" +
            "THREE[stop][stop]";
char[] charSeparators = new char[] { ',' };
string[] stringSeparators = new string[] { "[stop]" };
string[] result;
// ----------------------------------------------------------------------
-----
// Split a string delimited by characters.
// ----------------------------------------------------------------------
-----
Console.WriteLine("1) Split a string delimited by characters:\n");

// Display the original string and delimiter characters.
Console.WriteLine($"1a) The original string is \"{s1}\".");
Console.WriteLine($"The delimiter character is '{charSeparators[0]}'.\n");

// Split a string delimited by characters and return all elements.
Console.WriteLine("1b) Split a string delimited by characters and " +
                  "return all elements:");
result = s1.Split(charSeparators, StringSplitOptions.None);
Show(result);

// Split a string delimited by characters and return all non-empty elements.
Console.WriteLine("1c) Split a string delimited by characters and " +
                  "return all non-empty elements:");
result = s1.Split(charSeparators, StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the string and empty string before the
// delimiter and the remainder of the original string after the delimiter.
Console.WriteLine("1d) Split a string delimited by characters and " +
                  "return 2 elements:");
result = s1.Split(charSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("1e) Split a string delimited by characters and " +
                  "return 2 non-empty elements:");
result = s1.Split(charSeparators, 2, StringSplitOptions.RemoveEmptyEntries);
Show(result);
```

```
// ---------------------------------------------------------------------------
// Split a string delimited by another string.
// ---------------------------------------------------------------------------
Console.WriteLine("2) Split a string delimited by another string:\n");

// Display the original string and delimiter string.
Console.WriteLine($"2a) The original string is \"{s2}\".");
Console.WriteLine($"The delimiter string is \"{stringSeparators[0]}\".\n");

// Split a string delimited by another string and return all elements.
Console.WriteLine("2b) Split a string delimited by another string and " +
                  "return all elements:");
result = s2.Split(stringSeparators, StringSplitOptions.None);
Show(result);

// Split the original string at the delimiter and return all non-empty
elements.
Console.WriteLine("2c) Split a string delimited by another string and " +
                  "return all non-empty elements:");
result = s2.Split(stringSeparators, StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Split the original string into the empty string before the
// delimiter and the remainder of the original string after the delimiter.
Console.WriteLine("2d) Split a string delimited by another string and " +
                  "return 2 elements:");
result = s2.Split(stringSeparators, 2, StringSplitOptions.None);
Show(result);

// Split the original string into the string after the delimiter and the
// remainder of the original string after the delimiter.
Console.WriteLine("2e) Split a string delimited by another string and " +
                  "return 2 non-empty elements:");
result = s2.Split(stringSeparators, 2,
StringSplitOptions.RemoveEmptyEntries);
Show(result);

// Display the array of separated strings using a local function
void Show(string[] entries)
{
    Console.WriteLine($"The return value contains these {entries.Length}
elements:");
    foreach (string entry in entries)
    {
        Console.Write($"<{entry}>");
    }
    Console.Write("\n\n");
}

/*
This example produces the following results:
```

```
1) Split a string delimited by characters:

1a) The original string is ",ONE,,TWO,,,THREE,,".
The delimiter character is ',',

1b) Split a string delimited by characters and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

1c) Split a string delimited by characters and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

1d) Split a string delimited by characters and return 2 elements:
The return value contains these 2 elements:
<><ONE,,TWO,,,THREE,,>

1e) Split a string delimited by characters and return 2 non-empty elements:
The return value contains these 2 elements:
<ONE><TWO,,,THREE,,>

2) Split a string delimited by another string:

2a) The original string is "[stop]ONE[stop][stop]TWO[stop][stop]
[stop]THREE[stop][stop]".
The delimiter string is "[stop]".

2b) Split a string delimited by another string and return all elements:
The return value contains these 9 elements:
<><ONE><><TWO><><><THREE><><>

2c) Split a string delimited by another string and return all non-empty
elements:
The return value contains these 3 elements:
<ONE><TWO><THREE>

2d) Split a string delimited by another string and return 2 elements:
The return value contains these 2 elements:
<><ONE[stop][stop]TWO[stop][stop][stop]THREE[stop][stop]>

2e) Split a string delimited by another string and return 2 non-empty
elements:
The return value contains these 2 elements:
<ONE><TWO[stop][stop][stop]THREE[stop][stop]>

*/
```

Remarks

The String.Split method returns an array of the substrings in a given string that are delimited by specified characters or strings. Adjacent delimiters yield an array element

that contains an empty string (`""`). One of the fields of the `StringSplitOptions` enumeration specifies whether an element that contains an empty string should be included in the returned array.

Specify the `None` field to invoke the default behavior of the String.Split method, which is not to trim white-space characters and to include empty substrings.

The `TrimEntries` field is only available in .NET 5 and later versions.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Empty

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SystemException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as the base class for system exceptions namespace.

C#

```
public class SystemException : Exception
```

Inheritance Object → Exception → SystemException

Derived System.AccessViolationException
System.Activities.ValidationException
System.AppDomainUnloadedException
System.ArgumentException
System.ArithmeticException
More...

Remarks

This class is provided as a means to differentiate between system exceptions and application exceptions. It is the base class of such exceptions as ArgumentException, FormatException, and InvalidOperationException.

> ⓘ **Important**
>
> Because **SystemException** serves as the base class of a variety of exception types, your code should not throw a **SystemException** exception, nor should it attempt to handle a **SystemException** exception unless you intend to re-throw the original exception.

SystemException uses the HRESULT COR_E_SYSTEM, that has the value 0x80131501.

Constructors

Expand table

SystemException()	Initializes a new instance of the SystemException class.
SystemException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the SystemException class with serialized data.
SystemException(String)	Initializes a new instance of the SystemException class with a specified error message.
SystemException(String, Exception)	Initializes a new instance of the SystemException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- ApplicationException
- Handling and Throwing Exceptions
- Using Standard Exception Types

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

SystemException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the SystemException class.

Overloads

Expand table

SystemException()	Initializes a new instance of the SystemException class.
SystemException(String)	Initializes a new instance of the SystemException class with a specified error message.
SystemException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the SystemException class with serialized data.
SystemException(String, Exception)	Initializes a new instance of the SystemException class with a specified error message and a reference to the inner exception that is the cause of this exception.

SystemException()

Initializes a new instance of the SystemException class.

C#

```
public SystemException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "A system error has occurred."

This message takes into account the current system culture.

The following table shows the initial property values for an instance of SystemException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SystemException(String)

Initializes a new instance of the SystemException class with a specified error message.

C#

```
public SystemException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of SystemException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SystemException(SerializationInfo, StreamingContext)

Caution

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the SystemException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
```

```
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected SystemException (System.Runtime.Serialization.SerializationInfo
info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

SystemException(String, Exception)

Initializes a new instance of the SystemException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#
public SystemException (string? message, Exception? innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of SystemException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ThreadStaticAttribute Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates that the value of a static field is unique for each thread.

C#

```
[System.AttributeUsage(System.AttributeTargets.Field, Inherited=false)]
public class ThreadStaticAttribute : Attribute
```

Inheritance Object → Attribute → ThreadStaticAttribute

Attributes AttributeUsageAttribute

Examples

The following example instantiates a random number generator, creates ten threads in addition to the main thread, and then generates two million random numbers in each thread. It uses the ThreadStaticAttribute attribute to calculate the sum and the count of random numbers per thread. It also defines two additional per-thread fields, `previous` and `abnormal`, that allows it to detect corruption of the random number generator.

C#

```
using System;
using System.Threading;

public class Example
{
   [ThreadStatic] static double previous = 0.0;
   [ThreadStatic] static double sum = 0.0;
   [ThreadStatic] static int calls = 0;
   [ThreadStatic] static bool abnormal;
   static int totalNumbers = 0;
   static CountdownEvent countdown;
   private static Object lockObj;
   Random rand;
```

```
   public Example()
   {
      rand = new Random();
      lockObj = new Object();
      countdown = new CountdownEvent(1);
   }

   public static void Main()
   {
      Example ex = new Example();
      Thread.CurrentThread.Name = "Main";
      ex.Execute();
      countdown.Wait();
      Console.WriteLine("{0:N0} random numbers were generated.",
totalNumbers);
   }

   private void Execute()
   {
      for (int threads = 1; threads <= 10; threads++)
      {
         Thread newThread = new Thread(new
ThreadStart(this.GetRandomNumbers));
         countdown.AddCount();
         newThread.Name = threads.ToString();
         newThread.Start();
      }
      this.GetRandomNumbers();
   }

   private void GetRandomNumbers()
   {
      double result = 0.0;

      for (int ctr = 0; ctr < 2000000; ctr++)
      {
         lock (lockObj) {
            result = rand.NextDouble();
            calls++;
            Interlocked.Increment(ref totalNumbers);
            // We should never get the same random number twice.
            if (result == previous) {
               abnormal = true;
               break;
            }
            else {
               previous = result;
               sum += result;
            }
         }
      }
      // get last result
      if (abnormal)
         Console.WriteLine("Result is {0} in {1}", previous,
Thread.CurrentThread.Name);
```

```
      Console.WriteLine("Thread {0} finished random number generation.",
Thread.CurrentThread.Name);
      Console.WriteLine("Sum = {0:N4}, Mean = {1:N4}, n = {2:N0}\n", sum,
sum/calls, calls);
      countdown.Signal();
   }
}
// The example displays output similar to the following:
//    Thread 1 finished random number generation.
//    Sum = 1,000,556.7483, Mean = 0.5003, n = 2,000,000
//
//    Thread 6 finished random number generation.
//    Sum = 999,704.3865, Mean = 0.4999, n = 2,000,000
//
//    Thread 2 finished random number generation.
//    Sum = 999,680.8904, Mean = 0.4998, n = 2,000,000
//
//    Thread 10 finished random number generation.
//    Sum = 999,437.5132, Mean = 0.4997, n = 2,000,000
//
//    Thread 8 finished random number generation.
//    Sum = 1,000,663.7789, Mean = 0.5003, n = 2,000,000
//
//    Thread 4 finished random number generation.
//    Sum = 999,379.5978, Mean = 0.4997, n = 2,000,000
//
//    Thread 5 finished random number generation.
//    Sum = 1,000,011.0605, Mean = 0.5000, n = 2,000,000
//
//    Thread 9 finished random number generation.
//    Sum = 1,000,637.4556, Mean = 0.5003, n = 2,000,000
//
//    Thread Main finished random number generation.
//    Sum = 1,000,676.2381, Mean = 0.5003, n = 2,000,000
//
//    Thread 3 finished random number generation.
//    Sum = 999,951.1025, Mean = 0.5000, n = 2,000,000
//
//    Thread 7 finished random number generation.
//    Sum = 1,000,844.5217, Mean = 0.5004, n = 2,000,000
//
//    22,000,000 random numbers were generated.
```

The example uses the `lock` statement in C#, the `lock` function in F#, and the `SyncLock` construct in Visual Basic to synchronize access to the random number generator. This prevents corruption of the random number generator, which typically results in its returning a value of zero for all subsequent calls.

The example also uses the CountdownEvent class to ensure that each thread has finished generating random numbers before it displays the total number of calls.

Otherwise, if the main thread completes execution before the additional threads that it spawns, it displays an inaccurate value for the total number of method calls.

Remarks

A `static` field marked with ThreadStaticAttribute is not shared between threads. Each executing thread has a separate instance of the field, and independently sets and gets values for that field. If the field is accessed on a different thread, it will contain a different value.

Note that in addition to applying the ThreadStaticAttribute attribute to a field, you must also define it as a `static` field (in C# or F#) or a `Shared` field (in Visual Basic).

> **Note**
>
> Do not specify initial values for fields marked with `ThreadStaticAttribute`, because such initialization occurs only once, when the class constructor executes, and therefore affects only one thread. If you do not specify an initial value, you can rely on the field being initialized to its default value if it is a value type, or to `null` if it is a reference type.

Use this attribute as it is, and do not derive from it.

For more information about using attributes, see Attributes.

Constructors

Expand table

ThreadStaticAttribute()	Initializes a new instance of the ThreadStaticAttribute class.

Properties

Expand table

TypeId	When implemented in a derived class, gets a unique identifier for this Attribute. (Inherited from Attribute)

Methods

Expand table

Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object. (Inherited from Attribute)
GetHashCode()	Returns the hash code for this instance. (Inherited from Attribute)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsDefaultAttribute()	When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class. (Inherited from Attribute)
Match(Object)	When overridden in a derived class, returns a value that indicates whether this instance equals a specified object. (Inherited from Attribute)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Attribute
- Thread
- AsyncLocal<T>
- Extending Metadata Using Attributes

- Managed Threading

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ThreadStaticAttribute Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ThreadStaticAttribute class.

C#

```
public ThreadStaticAttribute ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a time of day, as would be read from a clock, within the range 00:00:00 to 23:59:59.9999999.

C#

```
public readonly struct TimeOnly : IComparable, IComparable<TimeOnly>,
IEquatable<TimeOnly>, IParsable<TimeOnly>, ISpanFormattable,
ISpanParsable<TimeOnly>, IUtf8SpanFormattable
```

Inheritance Object → ValueType → TimeOnly

Implements IComparable , IComparable<TimeOnly> , IEquatable<TimeOnly> , IFormattable , ISpanFormattable , IParsable<TimeOnly> , IParsable<TSelf> , ISpanParsable<TimeOnly> , IUtf8SpanFormattable

Constructors

Expand table

TimeOnly(Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour and the minute.
TimeOnly(Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, and second.
TimeOnly(Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, and millisecond.
TimeOnly(Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, millisecond, and microsecond.
TimeOnly(Int64)	Initializes a new instance of the TimeOnly structure using a specified number of ticks.

Properties

Expand table

Hour	Gets the hour component of the time represented by this instance.
MaxValue	Gets the largest possible value of TimeOnly.
Microsecond	Gets the microsecond component of the time represented by this instance.
Millisecond	Gets the millisecond component of the time represented by this instance.
Minute	Gets the minute component of the time represented by this instance.
MinValue	Gets the smallest possible value of TimeOnly.
Nanosecond	Gets the nanosecond component of the time represented by this instance.
Second	Gets the seconds component of the time represented by this instance.
Ticks	Gets the number of ticks that represent the time of this instance.

Methods

Expand table

Add(TimeSpan)	Returns a new TimeOnly that adds the value of the specified time span to the value of this instance.
Add(TimeSpan, Int32)	Returns a new TimeOnly that adds the value of the specified time span to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.
AddHours(Double)	Returns a new TimeOnly that adds the specified number of hours to the value of this instance.
AddHours(Double, Int32)	Returns a new TimeOnly that adds the specified number of hours to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.
AddMinutes(Double)	Returns a new TimeOnly that adds the specified number of minutes to the value of this instance.
AddMinutes(Double, Int32)	Returns a new TimeOnly that adds the specified number of minutes to the value of this instance. If the result wraps

	past the end of the day, this method returns the number of excess days as an out parameter.
CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified TimeOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.
CompareTo(TimeOnly)	Compares the value of this instance to a specified TimeOnly value and indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.
Deconstruct(Int32, Int32)	Deconstructs this TimeOnly instance into Hour and Minute.
Deconstruct(Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, and Second.
Deconstruct(Int32, Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, Second, and Millisecond.
Deconstruct(Int32, Int32, Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, Second, Millisecond, and Microsecond.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(TimeOnly)	Returns a value indicating whether the value of this instance is equal to the value of the specified TimeOnly instance.
FromDateTime(DateTime)	Constructs a TimeOnly object from a DateTime representing the time of the day in this DateTime object.
FromTimeSpan(TimeSpan)	Constructs a TimeOnly object from a time span representing the time elapsed since midnight.
GetHashCode()	Returns the hash code for this instance.
IsBetween(TimeOnly, TimeOnly)	Determines if a time falls within the range provided. Supports both "normal" ranges such as 10:00-12:00, and ranges that span midnight such as 23:00-01:00.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.

Parse(String)	Converts the string representation of a time to its TimeOnly equivalent by using the conventions of the current culture.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[])	Converts the specified span to its TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String[])	Converts the specified span to a TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ToLongTimeString()	Converts the value of the current TimeOnly instance to its equivalent long date string representation.

ToShortTimeString()	Converts the current TimeOnly instance to its equivalent short time string representation.
ToString()	Converts the current TimeOnly instance to its equivalent short time string representation using the formatting conventions of the current culture.
ToString(IFormatProvider)	Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the current TimeOnly instance to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString(String, IFormatProvider)	Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.
ToTimeSpan()	Convert the current TimeOnly instance to a TimeSpan object.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current TimeOnly instance into the provided span of characters.
TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeOnly)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, TimeOnly)	Tries to parse a string into a value.
TryParse(String, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], TimeOnly)	Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

Operators

Expand table

Equality(TimeOnly, TimeOnly)	Determines whether two specified instances of TimeOnlyare equal.

GreaterThan(TimeOnly, TimeOnly)	Determines whether one specified TimeOnly is later than another specified TimeOnly.
GreaterThanOrEqual(TimeOnly, TimeOnly)	Determines whether one specified TimeOnly represents a time that is the same as or later than another specified TimeOnly.
Inequality(TimeOnly, TimeOnly)	Determines whether two specified instances of TimeOnly are not equal.
LessThan(TimeOnly, TimeOnly)	Determines whether one specified TimeOnly is earlier than another specified TimeOnly.
LessThanOrEqual(TimeOnly, TimeOnly)	Determines whether one specified TimeOnly represents a time that is the same as or earlier than another specified TimeOnly.
Subtraction(TimeOnly, TimeOnly)	Gives the elapsed time between two points on a circular clock, which will always be a positive value.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TimeOnly(Int64)	Initializes a new instance of the TimeOnly structure using a specified number of ticks.
TimeOnly(Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour and the minute.
TimeOnly(Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, and second.
TimeOnly(Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, and millisecond.
TimeOnly(Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, millisecond, and microsecond.

TimeOnly(Int64)

Initializes a new instance of the TimeOnly structure using a specified number of ticks.

C#

```
public TimeOnly (long ticks);
```

Parameters

`ticks` Int64

A time of day expressed in the number of 100-nanosecond units since 00:00:00.0000000.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TimeOnly(Int32, Int32)

Initializes a new instance of the TimeOnly structure to the specified hour and the minute.

C#

```
public TimeOnly (int hour, int minute);
```

Parameters

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TimeOnly(Int32, Int32, Int32)

Initializes a new instance of the TimeOnly structure to the specified hour, minute, and second.

C#

```
public TimeOnly (int hour, int minute, int second);
```

Parameters

hour Int32

The hours (0 through 23).

minute Int32

The minutes (0 through 59).

second Int32

The seconds (0 through 59).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TimeOnly(Int32, Int32, Int32, Int32)

Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, and millisecond.

C#

```
public TimeOnly (int hour, int minute, int second, int millisecond);
```

Parameters

hour Int32

The hours (0 through 23).

minute Int32

The minutes (0 through 59).

second Int32

The seconds (0 through 59).

`millisecond` Int32

The millisecond (0 through 999).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TimeOnly(Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the TimeOnly structure to the specified hour, minute, second, millisecond, and microsecond.

```
C#

public TimeOnly (int hour, int minute, int second, int millisecond, int
microsecond);
```

Parameters

`hour` Int32

The hours (0 through 23).

`minute` Int32

The minutes (0 through 59).

`second` Int32

The seconds (0 through 59).

`millisecond` Int32

The millisecond (0 through 999).

`microsecond` Int32

The microsecond (0 through 999).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Hour Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hour component of the time represented by this instance.

C#

```
public int Hour { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the largest possible value of TimeOnly.

C#

```
public static TimeOnly MaxValue { get; }
```

Property Value

TimeOnly

The largest possible value of TimeOnly, which is 23:59:59.9999999.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Microsecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the microsecond component of the time represented by this instance.

C#

```
public int Microsecond { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Millisecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the millisecond component of the time represented by this instance.

C#

```
public int Millisecond { get; }
```

Property Value

Int32

A number representing the millisecond component of this TimeOnly.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Minute Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minute component of the time represented by this instance.

C#

```
public int Minute { get; }
```

Property Value

Int32

A number representing the minute component of this TimeOnly.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the smallest possible value of TimeOnly.

C#

```
public static TimeOnly MinValue { get; }
```

Property Value

TimeOnly

The smallest possible value of TimeOnly, which is 00:00:00 (midnight).

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Nanosecond Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the nanosecond component of the time represented by this instance.

C#

```
public int Nanosecond { get; }
```

Property Value

Int32

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Second Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the seconds component of the time represented by this instance.

C#

```
public int Second { get; }
```

Property Value

Int32

A number representing the seconds component of this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Ticks Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of ticks that represent the time of this instance.

C#

```
public long Ticks { get; }
```

Property Value

Int64

A number representing the number of ticks of this instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Add Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Add(TimeSpan)	Returns a new TimeOnly that adds the value of the specified time span to the value of this instance.
Add(TimeSpan, Int32)	Returns a new TimeOnly that adds the value of the specified time span to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.

Add(TimeSpan)

Returns a new TimeOnly that adds the value of the specified time span to the value of this instance.

C#

```
public TimeOnly Add (TimeSpan value);
```

Parameters

`value` TimeSpan

A positive or negative time interval.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the time interval represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Add(TimeSpan, Int32)

Returns a new TimeOnly that adds the value of the specified time span to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.

C#

```
public TimeOnly Add (TimeSpan value, out int wrappedDays);
```

Parameters

`value` TimeSpan

A positive or negative time interval.

`wrappedDays` Int32

When this method returns, contains the number of excess days, if any, that resulted from wrapping during this addition operation.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the time interval represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.AddHours Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

AddHours(Double, Int32)	Returns a new TimeOnly that adds the specified number of hours to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.
AddHours(Double)	Returns a new TimeOnly that adds the specified number of hours to the value of this instance.

AddHours(Double, Int32)

Returns a new TimeOnly that adds the specified number of hours to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.

C#

```
public TimeOnly AddHours (double value, out int wrappedDays);
```

Parameters

`value` Double

A number of whole and fractional hours. The value parameter can be negative or positive.

`wrappedDays` Int32

When this method returns, contains the number of excess days, if any, that resulted from wrapping during this addition operation.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the number of hours represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AddHours(Double)

Returns a new TimeOnly that adds the specified number of hours to the value of this instance.

```
C#

public TimeOnly AddHours (double value);
```

Parameters

`value` Double

A number of whole and fractional hours. The value parameter can be negative or positive.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the number of hours represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.AddMinutes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

AddMinutes(Double)	Returns a new TimeOnly that adds the specified number of minutes to the value of this instance.
AddMinutes(Double, Int32)	Returns a new TimeOnly that adds the specified number of minutes to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.

AddMinutes(Double)

Returns a new TimeOnly that adds the specified number of minutes to the value of this instance.

C#

```
public TimeOnly AddMinutes (double value);
```

Parameters

`value` Double

A number of whole and fractional minutes. The value parameter can be negative or positive.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the number of minutes represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

AddMinutes(Double, Int32)

Returns a new TimeOnly that adds the specified number of minutes to the value of this instance. If the result wraps past the end of the day, this method returns the number of excess days as an out parameter.

C#

```
public TimeOnly AddMinutes (double value, out int wrappedDays);
```

Parameters

`value` Double

A number of whole and fractional minutes. The value parameter can be negative or positive.

`wrappedDays` Int32

When this method returns, contains the number of excess days, if any, that resulted from wrapping during this addition operation.

Returns

TimeOnly

An object whose value is the sum of the time represented by this instance and the number of minutes represented by value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(Object)	Compares the value of this instance to a specified object that contains a specified TimeOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.
CompareTo(TimeOnly)	Compares the value of this instance to a specified TimeOnly value and indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.

CompareTo(Object)

Compares the value of this instance to a specified object that contains a specified TimeOnly value, and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

A boxed object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and the value parameter. Less than zero if this instance is earlier than value. Zero if this instance is the same as value. Greater than zero if this instance is later than value.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

CompareTo(TimeOnly)

Compares the value of this instance to a specified TimeOnly value and indicates whether this instance is earlier than, the same as, or later than the specified TimeOnly value.

```
C#

public int CompareTo (TimeOnly value);
```

Parameters

`value` TimeOnly

The object to compare to the current instance.

Returns

Int32

A signed number indicating the relative values of this instance and the value parameter.

- Less than zero if this instance is earlier than value.
- Zero if this instance is the same as value.

- Greater than zero if this instance is later than value.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Deconstruct Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Deconstruct(Int32, Int32, Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, Second, Millisecond, and Microsecond.
Deconstruct(Int32, Int32)	Deconstructs this TimeOnly instance into Hour and Minute.
Deconstruct(Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, and Second.
Deconstruct(Int32, Int32, Int32, Int32)	Deconstructs this TimeOnly instance into Hour, Minute, Second, and Millisecond.

Deconstruct(Int32, Int32, Int32, Int32, Int32)

Deconstructs this TimeOnly instance into Hour, Minute, Second, Millisecond, and Microsecond.

C#

```
public void Deconstruct (out int hour, out int minute, out int second,
out int millisecond, out int microsecond);
```

Parameters

hour Int32

When this method returns, contains the Hour value for this TimeOnly instance.

minute Int32

When this method returns, contains the Minute value for this TimeOnly instance.

`second` Int32

When this method returns, contains the Second value for this TimeOnly instance.

`millisecond` Int32

When this method returns, contains the Millisecond value for this TimeOnly instance.

`microsecond` Int32

When this method returns, contains the Microsecond value for this TimeOnly instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Deconstruct(Int32, Int32)

Deconstructs this TimeOnly instance into Hour and Minute.

C#

```
public void Deconstruct (out int hour, out int minute);
```

Parameters

`hour` Int32

When this method returns, contains the Hour value for this TimeOnly instance.

`minute` Int32

When this method returns, contains the Minute value for this TimeOnly instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Deconstruct(Int32, Int32, Int32)

Deconstructs this TimeOnly instance into Hour, Minute, and Second.

```
C#

public void Deconstruct (out int hour, out int minute, out int second);
```

Parameters

hour Int32

When this method returns, contains the Hour value for this TimeOnly instance.

minute Int32

When this method returns, contains the Minute value for this TimeOnly instance.

second Int32

When this method returns, contains the Second value for this TimeOnly instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Deconstruct(Int32, Int32, Int32, Int32)

Deconstructs this TimeOnly instance into Hour, Minute, Second, and Millisecond.

```
C#

public void Deconstruct (out int hour, out int minute, out int second,
out int millisecond);
```

Parameters

`hour` Int32

When this method returns, contains the Hour value for this TimeOnly instance.

`minute` Int32

When this method returns, contains the Minute value for this TimeOnly instance.

`second` Int32

When this method returns, contains the Second value for this TimeOnly instance.

`millisecond` Int32

When this method returns, contains the Millisecond value for this TimeOnly instance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(TimeOnly)	Returns a value indicating whether the value of this instance is equal to the value of the specified TimeOnly instance.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.

Equals(TimeOnly)

Returns a value indicating whether the value of this instance is equal to the value of the specified TimeOnly instance.

```
C#

public bool Equals (TimeOnly value);
```

Parameters

`value` TimeOnly

The object to compare to this instance.

Returns

Boolean

`true` if the value parameter equals the value of this instance; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

```
C#
public override bool Equals (object? value);
```

Parameters

`value` Object

The object to compare to this instance.

Returns

Boolean

`true` if value is an instance of TimeOnly and equals the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.FromDateTime(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Constructs a TimeOnly object from a DateTime representing the time of the day in this DateTime object.

```
C#
public static TimeOnly FromDateTime (DateTime dateTime);
```

Parameters

`dateTime` DateTime

The DateTime object to extract the time of the day from.

Returns

TimeOnly

A TimeOnly object representing time of the day specified in the DateTime object.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeOnly.FromTimeSpan(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Constructs a TimeOnly object from a time span representing the time elapsed since midnight.

C#

```
public static TimeOnly FromTimeSpan (TimeSpan timeSpan);
```

Parameters

`timeSpan` TimeSpan

The time interval measured since midnight. This value has to be positive and not exceeding the time of the day.

Returns

TimeOnly

A TimeOnly object representing the time elapsed since midnight using the specified time span value.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.IsBetween(TimeOnly, TimeOnly) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a time falls within the range provided. Supports both "normal" ranges such as 10:00-12:00, and ranges that span midnight such as 23:00-01:00.

```
C#
public bool IsBetween (TimeOnly start, TimeOnly end);
```

Parameters

`start` TimeOnly

The starting time of day, inclusive.

`end` TimeOnly

The ending time of day, exclusive.

Returns

Boolean

`true`, if the time falls within the range, `false` otherwise.

Remarks

If `start` and `end` are equal, this method returns `false`, meaning there is zero elapsed time between the two values. If you wish to treat such cases as representing one or more whole days, then first check for equality before calling this method.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String)	Converts the string representation of a time to its TimeOnly equivalent by using the conventions of the current culture.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts a memory span that contains string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.
Parse(String, IFormatProvider, DateTimeStyles)	Converts the string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.

Parse(String)

Converts the string representation of a time to its TimeOnly equivalent by using the conventions of the current culture.

C#

```
public static TimeOnly Parse (string s);
```

Parameters

`s` String

The string to parse.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static TimeOnly Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

A span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

TimeOnly

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, IFormatProvider)

Parses a string into a value.

```
C#

public static TimeOnly Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

TimeOnly

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts a memory span that contains string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.

C#

```
public static TimeOnly Parse (ReadOnlySpan<char> s, IFormatProvider?
provider = default, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

The memory span that contains the time to parse.

`provider` IFormatProvider

The culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(String, IFormatProvider, DateTimeStyles)

Converts the string representation of a time to its TimeOnly equivalent by using culture-specific format information and a formatting style.

C#

```
public static TimeOnly Parse (string s, IFormatProvider? provider,
System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

`s` String

The string containing the time to parse.

`provider` IFormatProvider

The culture-specific format information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in s for the parse operation to succeed, and that defines how to interpret the parsed date. A typical value to specify is None.

Returns

TimeOnly

ATimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ParseExact(String, String[], IFormatProvider, DateTimeStyles)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(String, String, IFormatProvider, DateTimeStyles)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(String, String[])	Converts the specified span to a TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

ParseExact(String, String)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format. The format of the string representation must match the specified format exactly or an exception is thrown.
ParseExact(ReadOnlySpan<Char>, String[])	Converts the specified span to its TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

ParseExact(String, String[], IFormatProvider, DateTimeStyles)

Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (string s, string[] formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s String

A string containing the time to convert.

formats String[]

An array of allowable formats of `s`.

provider IFormatProvider

The culture-specific formatting information about `s`.

style DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String, IFormatProvider, DateTimeStyles)

Converts the specified string representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (string s, string format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s String

A string containing the time to convert.

format String

The format specifier that defines the required format of `s`.

`provider` IFormatProvider

The culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of the enumeration values that provides additional information about `s`, about style elements that may be present in `s`, or about the conversion from `s` to a TimeOnly value. A typical value to specify is None.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information, and style. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (ReadOnlySpan<char> s, string[]
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style = System.Globalization.DateTimeStyles.None);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the time to convert.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

The culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider = default,
System.Globalization.DateTimeStyles style =
System.Globalization.DateTimeStyles.None);
```

Parameters

s ReadOnlySpan<Char>

A span containing the time to convert.

format ReadOnlySpan<Char>

The format specifier that defines the required format of `s`.

provider IFormatProvider

The culture-specific formatting information about `s`.

style DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String[])

Converts the specified span to a TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (string s, string[] formats);
```

Parameters

`s` String

A span containing the time to convert.

`formats` String[]

An array of allowable formats of `s`.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(String, String)

Converts the specified string representation of a time to its TimeOnly equivalent using the specified format. The format of the string representation must match the specified format exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (string s, string format);
```

Parameters

`s` String

A string containing a time to convert.

`format` String

A format specifier that defines the required format of `s`.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ParseExact(ReadOnlySpan<Char>, String[])

Converts the specified span to its TimeOnly equivalent using the specified array of formats. The format of the string representation must match at least one of the specified formats exactly or an exception is thrown.

C#

```
public static TimeOnly ParseExact (ReadOnlySpan<char> s, string[]
formats);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the time to convert.

`formats` String[]

An array of allowable formats of `s`.

Returns

TimeOnly

A TimeOnly instance.

Exceptions

FormatException

`s` does not contain a valid string representation of a time.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.ToLongTimeString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current TimeOnly instance to its equivalent long date string representation.

C#

```
public string ToLongTimeString ();
```

Returns

String

The long time string representation of the current instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.ToShortTimeString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current TimeOnly instance to its equivalent short time string representation.

C#

```
public string ToShortTimeString ();
```

Returns

String

The short time string representation of the current instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString(String, IFormatProvider)	Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the current TimeOnly instance to its equivalent string representation using the specified format and the formatting conventions of the current culture.
ToString()	Converts the current TimeOnly instance to its equivalent short time string representation using the formatting conventions of the current culture.
ToString(IFormatProvider)	Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.

ToString(String, IFormatProvider)

Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

format String

A standard or custom time format string.

`provider` IFormatProvider

The culture-specific formatting information.

Returns

String

A string representation of value of the current instance.

Implements

ToString(String, IFormatProvider)

Remarks

The accepted standard formats are 'r', 'R', 'o', 'O', 't' and 'T'.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString(String)

Converts the current TimeOnly instance to its equivalent string representation using the specified format and the formatting conventions of the current culture.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom time format string.

Returns

String

A string representation of the current instance with the specified format and the formatting conventions of the current culture.

Remarks

The accepted standard formats are 'r', 'R', 'o', 'O', 't' and 'T'.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString()

Converts the current TimeOnly instance to its equivalent short time string representation using the formatting conventions of the current culture.

C#

```
public override string ToString ();
```

Returns

String

The short time string representation of the current instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ToString(IFormatProvider)

Converts the value of the current TimeOnly instance to its equivalent string representation using the specified culture-specific format information.

```
C#

public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

The culture-specific formatting information.

Returns

String

A string representation of the current instance as specified by the provider.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.ToTimeSpan Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Convert the current TimeOnly instance to a TimeSpan object.

C#

```
public TimeSpan ToTimeSpan ();
```

Returns

TimeSpan

A TimeSpan object spanning to the time specified in the current TimeOnly instance.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current TimeOnly instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current TimeOnly instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Remarks

The accepted standard formats are 'r', 'R', 'o', 'O', 't', and 'T'.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeOnly)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, TimeOnly)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.

TryParse(ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.

```C#
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, System.Globalization.DateTimeStyles style, out TimeOnly
result);
```

Parameters

`s` ReadOnlySpan<Char>

A string containing the characters that represent a time to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a date. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(ReadOnlySpan<Char>, TimeOnly)

Converts the specified span representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out TimeOnly result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the time to convert.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or MinValue if the conversion failed. The conversion fails if `s` is the empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if the conversion was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(String, TimeOnly)

Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse (string? s, out TimeOnly result);
```

Parameters

s String

A string containing the time to convert.

result TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeOnly)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out TimeOnly result);
```

Parameters

s ReadOnlySpan<Char>

A span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or MinValue if the conversion failed. The conversion fails if `s` is the empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, TimeOnly)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider, out
TimeOnly result);
```

Parameters

`s` String

A string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or MinValue if the conversion failed. The conversion fails if `s` is the empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was parsed successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified string representation of a time to its TimeOnly equivalent using the specified array of formats, culture-specific format information and style, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider,
System.Globalization.DateTimeStyles style, out TimeOnly result);
```

Parameters

`s` String

A string containing the time to convert.

`provider` IFormatProvider

The culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is None.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeOnly.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, String[], TimeOnly)	Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, TimeOnly)	Converts the specified span representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified

	format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.
TryParseExact(String, String[], IFormatProvider, DateTimeStyles, TimeOnly)	Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s, string?[]?
formats, IFormatProvider? provider, System.Globalization.DateTimeStyles
style, out TimeOnly result);
```

Parameters

s ReadOnlySpan<Char>

The span containing the time to parse.

formats String[]

An array of allowable formats of `s`.

provider IFormatProvider

An object that supplies culture-specific formatting information about `s`.

style DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed time. A typical value to specify is None.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, IFormatProvider? provider,
System.Globalization.DateTimeStyles style, out TimeOnly result);
```

Parameters

s ReadOnlySpan<Char>

A span containing the time to convert.

`format` ReadOnlySpan<Char>

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a time that corresponds to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String[], TimeOnly)

Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string?[]? formats, out
TimeOnly result);
```

Parameters

`s` String

The string containing the time to parse.

`formats` String[]

An array of allowable formats of `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String, IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified format, culture-specific format information, and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParseExact (string? s, string? format,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
TimeOnly result);
```

Parameters

`s` String

A span containing the characters representing a time to convert.

`format` String

The required format of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of one or more enumeration values that indicate the permitted format of `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a time that corresponds to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, String[], TimeOnly)

Converts the specified character span of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParseExact (ReadOnlySpan<char> s, string?[]?
formats, out TimeOnly result);
```

Parameters

s ReadOnlySpan<Char>

The span containing the time to convert.

formats String[]

An array of allowable formats of `s`.

result TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if the `s` parameter is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, TimeOnly)

Converts the specified span representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParseExact (ReadOnlySpan<char> s,
ReadOnlySpan<char> format, out TimeOnly result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the time to convert.

`format` ReadOnlySpan<Char>

The required format of `s`.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a time that corresponds to the pattern specified in `format`. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String, TimeOnly)

Converts the specified string representation of a time to its TimeOnly equivalent using the specified format and style. The format of the string representation must match the specified format exactly. The method returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParseExact (string? s, string? format, out TimeOnly
result);
```

Parameters

s String

A string containing the time to convert.

format String

The required format of `s`.

result TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a time that corresponds to the pattern specified in format. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParseExact(String, String[], IFormatProvider, DateTimeStyles, TimeOnly)

Converts the specified string representation of a time to its TimeOnly equivalent and returns a value that indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParseExact (string? s, string?[]? formats,
IFormatProvider? provider, System.Globalization.DateTimeStyles style, out
TimeOnly result);
```

Parameters

`s` String

The string containing the time to parse.

`formats` String[]

An array of allowable formats of `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`style` DateTimeStyles

A bitwise combination of enumeration values that defines how to interpret the parsed date. A typical value to specify is None.

`result` TimeOnly

When this method returns, contains the TimeOnly value equivalent to the time contained in `s`, if the conversion succeeded, or TimeOnly.MinValue if the conversion failed. The conversion fails if `s` is an empty string or does not contain a valid string representation of a time. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.Equality(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of TimeOnlyare equal.

C#

```
public static bool operator == (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left and right represent the same time; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.GreaterThan(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified TimeOnly is later than another specified TimeOnly.

```
C#
public static bool operator > (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left is later than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.GreaterThanOrEqual(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified TimeOnly represents a time that is the same as or later than another specified TimeOnly.

C#

```
public static bool operator >= (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left is the same as or later than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.Inequality(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of TimeOnly are not equal.

C#

```
public static bool operator != (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left and right do not represent the same time; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.LessThan(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified TimeOnly is earlier than another specified TimeOnly.

C#

```
public static bool operator < (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left is earlier than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.LessThanOrEqual(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether one specified TimeOnly represents a time that is the same as or earlier than another specified TimeOnly.

C#

```
public static bool operator <= (TimeOnly left, TimeOnly right);
```

Parameters

`left` TimeOnly

The first object to compare.

`right` TimeOnly

The second object to compare.

Returns

Boolean

`true` if left is the same as or earlier than right; otherwise, `false`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeOnly.Subtraction(TimeOnly, TimeOnly) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gives the elapsed time between two points on a circular clock, which will always be a positive value.

C#

```
public static TimeSpan operator - (TimeOnly t1, TimeOnly t2);
```

Parameters

`t1` TimeOnly

The first TimeOnly instance.

`t2` TimeOnly

The second TimeOnly instance..

Returns

TimeSpan

The elapsed time between `t1` and `t2`.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeoutException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the time allotted for a process or operation has expired.

C#

```
public class TimeoutException : SystemException
```

Inheritance Object → Exception → SystemException → TimeoutException

Derived System.Text.RegularExpressions.RegexMatchTimeoutException

Examples

The following code example demonstrates the use of TimeoutException in conjunction with members of the System.IO.Ports.SerialPort class.

C#

```
// This example demonstrates the use of the TimeoutException
// exception in conjunction with the SerialPort class.

using System;
using System.IO.Ports;

class Sample
{
    public static void Main()
    {
    string input;
    try
        {
// Set the COM1 serial port to speed = 4800 baud, parity = odd,
// data bits = 8, stop bits = 1.
        SerialPort sp = new SerialPort("COM1",
                        4800, Parity.Odd, 8, StopBits.One);
// Timeout after 2 seconds.
```

```
        sp.ReadTimeout = 2000;
        sp.Open();

// Read until either the default newline termination string
// is detected or the read operation times out.
        input = sp.ReadLine();

        sp.Close();

// Echo the input.
        Console.WriteLine(input);
        }

// Only catch timeout exceptions.
    catch (TimeoutException e)
        {
        Console.WriteLine(e);
        }
    }
}
/*
This example produces the following results:

(Data received at the serial port is echoed to the console if the
read operation completes successfully before the specified timeout period
expires. Otherwise, a timeout exception like the following is thrown.)

System.TimeoutException: The operation has timed-out.
   at System.IO.Ports.SerialStream.ReadByte(Int32 timeout)
   at System.IO.Ports.SerialPort.ReadOneChar(Int32 timeout)
   at System.IO.Ports.SerialPort.ReadTo(String value)
   at System.IO.Ports.SerialPort.ReadLine()
   at Sample.Main()
*/
```

Remarks

For information about setting the time-out interval before a TimeoutException exception is thrown, consult the documentation for the type whose method timed out. For information about handling the time-out, see the documentation for the method that threw the exception.

TimeoutException uses the HRESULT, COR_E_TIMEOUT, which has the value 0x80131505.

For a list of initial property values for an instance of TimeoutException, see the TimeoutException constructors.

Constructors

Expand table

TimeoutException()	Initializes a new instance of the TimeoutException class.
TimeoutException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TimeoutException class with serialized data.
TimeoutException(String)	Initializes a new instance of the TimeoutException class with the specified error message.
TimeoutException(String, Exception)	Initializes a new instance of the TimeoutException class with the specified error message and inner exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeoutException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TimeoutException class.

Overloads

Expand table

TimeoutException()	Initializes a new instance of the TimeoutException class.
TimeoutException(String)	Initializes a new instance of the TimeoutException class with the specified error message.
TimeoutException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TimeoutException class with serialized data.
TimeoutException(String, Exception)	Initializes a new instance of the TimeoutException class with the specified error message and inner exception.

TimeoutException()

Initializes a new instance of the TimeoutException class.

C#

```
public TimeoutException ();
```

Remarks

This constructor initializes the inherited Exception.Message property of the new instance with a system-supplied error message. This message takes into account the

current system culture.

The following table shows the initial property values for an instance of TimeoutException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeoutException(String)

Initializes a new instance of the TimeoutException class with the specified error message.

C#

```
public TimeoutException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

This constructor initializes the inherited Exception.Message property of the new instance using the value of the `message` parameter.

The following table shows the initial property values for an instance of TimeoutException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeoutException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the TimeoutException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected TimeoutException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo object that contains serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext object that contains contextual information about the source or destination. The `context` parameter is reserved for future use, and can be specified as `null`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` parameter is `null`.

SerializationException

The class name is `null`, or HResult is zero (0).

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see Binary Serialization.

See also

- SerializationInfo
- StreamingContext
- Binary Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TimeoutException(String, Exception)

Initializes a new instance of the TimeoutException class with the specified error message and inner exception.

C#

```
public TimeoutException (string? message, Exception? innerException);
```

Parameters

`message` String

The message that describes the error.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

This constructor initializes the inherited Exception.Message property of the new instance using the value of the `message` parameter.

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed to the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of TimeoutException.

Expand table

Property	Value
Message	The error message string.
InnerException	The inner exception reference.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeProvider Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an abstraction for time.

C#

```
public abstract class TimeProvider
```

Inheritance Object → TimeProvider

Derived Microsoft.Extensions.Time.Testing.FakeTimeProvider

Constructors

Expand table

TimeProvider()	Initializes a new instance of TimeProvider.

Properties

Expand table

LocalTimeZone	Gets the local time zone according to this TimeProvider's notion of time.
System	Gets a TimeProvider that provides a clock based on UtcNow, a time zone based on Local, a high-performance time stamp based on Stopwatch, and a timer based on Timer.
TimestampFrequency	Gets the frequency of GetTimestamp() as the number of ticks per second.

Methods

Expand table

CreateTimer(TimerCallback, Object, TimeSpan, TimeSpan)	Creates a new ITimer instance, using TimeSpan values to measure time intervals.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetElapsedTime(Int64)	Gets the elapsed time since the `startingTimestamp` value was retrieved using GetTimestamp().
GetElapsedTime(Int64, Int64)	Gets the elapsed time between two timestamps retrieved using GetTimestamp().
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetLocalNow()	Gets the current date and time according to this TimeProvider's notion of time based on GetUtcNow(), with the offset set to the LocalTimeZone's offset from Coordinated Universal Time (UTC).
GetTimestamp()	Gets the current high-frequency value designed to measure small time intervals with high accuracy in the timer mechanism.
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetUtcNow()	Gets the current Coordinated Universal Time (UTC) date and time with an offset of zero, according to this TimeProvider's notion of time.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeProvider Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of TimeProvider.

C#

```
protected TimeProvider ();
```

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeProvider.LocalTimeZone Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the local time zone according to this TimeProvider's notion of time.

C#

```
public virtual TimeZoneInfo LocalTimeZone { get; }
```

Property Value

TimeZoneInfo

Remarks

The default implementation returns Local.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeProvider.System Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a TimeProvider that provides a clock based on UtcNow, a time zone based on Local, a high-performance time stamp based on Stopwatch, and a timer based on Timer.

C#

```
public static TimeProvider System { get; }
```

Property Value

TimeProvider

Remarks

If the Local changes after the object is returned, the change will be reflected in any subsequent operations that retrieve GetLocalNow().

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeProvider.TimestampFrequency Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the frequency of GetTimestamp() as the number of ticks per second.

C#

```
public virtual long TimestampFrequency { get; }
```

Property Value

Int64

Remarks

The default implementation returns Frequency. For a given TimeProvider instance, the value must be idempotent and remain unchanged.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeProvider.CreateTimer(TimerCallback, Object, TimeSpan, TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new ITimer instance, using TimeSpan values to measure time intervals.

C#

```
public virtual System.Threading.ITimer CreateTimer
(System.Threading.TimerCallback callback, object? state, TimeSpan dueTime,
TimeSpan period);
```

Parameters

`callback` TimerCallback

A delegate representing a method to be executed when the timer fires. The method should be reentrant, as it may be invoked simultaneously on two threads if the timer fires again while a previous callback is still being handled.

`state` Object

An object to be passed to the `callback`. This may be `null`.

`dueTime` TimeSpan

The amount of time to delay before `callback` is invoked. Specify InfiniteTimeSpan to prevent the timer from starting. Specify Zero to start the timer immediately.

`period` TimeSpan

The time interval between invocations of `callback`. Specify InfiniteTimeSpan to disable periodic signaling.

Returns

ITimer

The newly created ITimer instance.

Exceptions

ArgumentNullException

`callback` is `null`.

ArgumentOutOfRangeException

The number of milliseconds in the value of `dueTime` or `period` is negative and not equal to Infinite, or is greater than Int32.MaxValue.

Remarks

The delegate specified by the callback parameter is invoked once after `dueTime` elapses, and thereafter each time the `period` time interval elapses.

If `dueTime` is zero, the callback is invoked immediately. If `dueTime` is -1 milliseconds, `callback` is not invoked; the timer is disabled, but can be re-enabled by calling the Change(TimeSpan, TimeSpan) method.

If `period` is 0 or -1 milliseconds and `dueTime` is positive, `callback` is invoked once; the periodic behavior of the timer is disabled, but can be re-enabled using the Change(TimeSpan, TimeSpan) method.

The returned ITimer instance is implicitly rooted while the timer is still scheduled.

CreateTimer(TimerCallback, Object, TimeSpan, TimeSpan) captures the ExecutionContext and stores that with the ITimer for use in invoking `callback` each time it's called. That capture can be suppressed with SuppressFlow().

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeProvider.GetElapsedTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetElapsedTime(Int64)	Gets the elapsed time since the `startingTimestamp` value was retrieved using GetTimestamp().
GetElapsedTime(Int64, Int64)	Gets the elapsed time between two timestamps retrieved using GetTimestamp().

GetElapsedTime(Int64)

Gets the elapsed time since the `startingTimestamp` value was retrieved using GetTimestamp().

C#

```
public TimeSpan GetElapsedTime (long startingTimestamp);
```

Parameters

`startingTimestamp` Int64

The timestamp marking the beginning of the time period.

Returns

TimeSpan

The elapsed time between the starting timestamp and the time of this call.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

GetElapsedTime(Int64, Int64)

Gets the elapsed time between two timestamps retrieved using GetTimestamp().

C#

```
public TimeSpan GetElapsedTime (long startingTimestamp, long
endingTimestamp);
```

Parameters

`startingTimestamp` Int64

The timestamp marking the beginning of the time period.

`endingTimestamp` Int64

The timestamp marking the end of the time period.

Returns

TimeSpan

The elapsed time between the starting and ending timestamps.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeProvider.GetLocalNow Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current date and time according to this TimeProvider's notion of time based on GetUtcNow(), with the offset set to the LocalTimeZone's offset from Coordinated Universal Time (UTC).

C#

```
public DateTimeOffset GetLocalNow ();
```

Returns

DateTimeOffset

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeProvider.GetTimestamp Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current high-frequency value designed to measure small time intervals with high accuracy in the timer mechanism.

C#

```
public virtual long GetTimestamp ();
```

Returns

Int64

A long integer representing the high-frequency counter value of the underlying timer mechanism.

Remarks

The default implementation returns GetTimestamp().

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeProvider.GetUtcNow Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current Coordinated Universal Time (UTC) date and time with an offset of zero, according to this TimeProvider's notion of time.

C#

```
public virtual DateTimeOffset GetUtcNow ();
```

Returns

DateTimeOffset

Remarks

The default implementation returns UtcNow.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a time interval.

C#

```
public readonly struct TimeSpan : IComparable, IComparable<TimeSpan>,
IEquatable<TimeSpan>, IParsable<TimeSpan>, ISpanFormattable,
ISpanParsable<TimeSpan>, IUtf8SpanFormattable
```

Inheritance Object → ValueType → TimeSpan

Implements IComparable , IComparable<TimeSpan> , IEquatable<TimeSpan> , IFormattable , ISpanFormattable , IParsable<TimeSpan> , IParsable<TSelf> , ISpanParsable<TimeSpan> , IUtf8SpanFormattable

Examples

The following example instantiates a TimeSpan object that represents the difference between two dates. It then displays the TimeSpan object's properties.

C#

```
// Define two dates.
DateTime date1 = new DateTime(2010, 1, 1, 8, 0, 15);
DateTime date2 = new DateTime(2010, 8, 18, 13, 30, 30);

// Calculate the interval between the two dates.
TimeSpan interval = date2 - date1;
Console.WriteLine("{0} - {1} = {2}", date2, date1, interval.ToString());

// Display individual properties of the resulting TimeSpan object.
Console.WriteLine("   {0,-35} {1,20}", "Value of Days Component:",
interval.Days);
Console.WriteLine("   {0,-35} {1,20}", "Total Number of Days:",
interval.TotalDays);
Console.WriteLine("   {0,-35} {1,20}", "Value of Hours Component:",
```

```
interval.Hours);
Console.WriteLine("   {0,-35} {1,20}", "Total Number of Hours:",
interval.TotalHours);
Console.WriteLine("   {0,-35} {1,20}", "Value of Minutes Component:",
interval.Minutes);
Console.WriteLine("   {0,-35} {1,20}", "Total Number of Minutes:",
interval.TotalMinutes);
Console.WriteLine("   {0,-35} {1,20:N0}", "Value of Seconds Component:",
interval.Seconds);
Console.WriteLine("   {0,-35} {1,20:N0}", "Total Number of Seconds:",
interval.TotalSeconds);
Console.WriteLine("   {0,-35} {1,20:N0}", "Value of Milliseconds
Component:", interval.Milliseconds);
Console.WriteLine("   {0,-35} {1,20:N0}", "Total Number of Milliseconds:",
interval.TotalMilliseconds);
Console.WriteLine("   {0,-35} {1,20:N0}", "Ticks:", interval.Ticks);

// This example displays the following output:
//       8/18/2010 1:30:30 PM - 1/1/2010 8:00:15 AM = 229.05:30:15
//          Value of Days Component:                              229
//          Total Number of Days:                   229.229340277778
//          Value of Hours Component:                               5
//          Total Number of Hours:                  5501.50416666667
//          Value of Minutes Component:                            30
//          Total Number of Minutes:                       330090.25
//          Value of Seconds Component:                            15
//          Total Number of Seconds:                       19,805,415
//          Value of Milliseconds Component:                        0
//          Total Number of Milliseconds:             19,805,415,000
//          Ticks:                                198,054,150,000,000
```

PowerShell

```
# Define two dates.
$Date2 = Get-Date -Date '2010/8/18' -Hour 13 -Minute 30 -Second 30
$Date1 = Get-Date -Date '2010/1/1'  -Hour 8  -Minute 0  -Second 15

# Calculate the interval between the two dates.
$Interval = $Date2 - $Date1
"{0} - {1} = {2}" -f $Date2, $Date1, ($Interval.ToString())

#  Display individual properties of the resulting TimeSpan object.
"   {0,-35} {1,20}"    -f "Value of Days Component:", $Interval.Days
"   {0,-35} {1,20}"    -f "Total Number of Days:", $Interval.TotalDays
"   {0,-35} {1,20}"    -f "Value of Hours Component:", $Interval.Hours
"   {0,-35} {1,20}"    -f "Total Number of Hours:", $Interval.TotalHours
"   {0,-35} {1,20}"    -f "Value of Minutes Component:", $Interval.Minutes
"   {0,-35} {1,20}"    -f "Total Number of Minutes:", $Interval.TotalMinutes
"   {0,-35} {1,20:N0}" -f "Value of Seconds Component:", $Interval.Seconds
"   {0,-35} {1,20:N0}" -f "Total Number of Seconds:", $Interval.TotalSeconds
"   {0,-35} {1,20:N0}" -f "Value of Milliseconds Component:",
$Interval.Milliseconds
"   {0,-35} {1,20:N0}" -f "Total Number of Milliseconds:",
```

```
$Interval.TotalMilliseconds
"   {0,-35} {1,20:N0}" -f "Ticks:", $Interval.Ticks

<# This sample produces the following output:

18/08/2010 13:30:30 - 01/01/2010 08:00:15 = 229.05:30:15
   Value of Days Component:                              229
   Total Number of Days:                   229.229340277778
   Value of Hours Component:                               5
   Total Number of Hours:                  5501.50416666667
   Value of Minutes Component:                            30
   Total Number of Minutes:                       330090.25
   Value of Seconds Component:                            15
   Total Number of Seconds:                      19,805,415
   Value of Milliseconds Component:                        0
   Total Number of Milliseconds:             19,805,415,000
   Ticks:                               198,054,150,000,000
#>
```

Remarks

For more information about this API, see Supplemental API remarks for TimeSpan.

Constructors

Expand table

TimeSpan(Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of hours, minutes, and seconds.
TimeSpan(Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, and seconds.
TimeSpan(Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, and milliseconds.
TimeSpan(Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, milliseconds, and microseconds.
TimeSpan(Int64)	Initializes a new instance of the TimeSpan structure to the specified number of ticks.

Fields

Expand table

MaxValue	Represents the maximum TimeSpan value. This field is read-only.
MinValue	Represents the minimum TimeSpan value. This field is read-only.
NanosecondsPerTick	Represents the number of nanoseconds per tick. This field is constant.
TicksPerDay	Represents the number of ticks in 1 day. This field is constant.
TicksPerHour	Represents the number of ticks in 1 hour. This field is constant.
TicksPerMicrosecond	Represents the number of ticks in 1 microsecond. This field is constant.
TicksPerMillisecond	Represents the number of ticks in 1 millisecond. This field is constant.
TicksPerMinute	Represents the number of ticks in 1 minute. This field is constant.
TicksPerSecond	Represents the number of ticks in 1 second.
Zero	Represents the zero TimeSpan value. This field is read-only.

Properties

Expand table

Days	Gets the days component of the time interval represented by the current TimeSpan structure.
Hours	Gets the hours component of the time interval represented by the current TimeSpan structure.
Microseconds	Gets the microseconds component of the time interval represented by the current TimeSpan structure.
Milliseconds	Gets the milliseconds component of the time interval represented by the current TimeSpan structure.
Minutes	Gets the minutes component of the time interval represented by the current TimeSpan structure.
Nanoseconds	Gets the nanoseconds component of the time interval represented by the current TimeSpan structure.
Seconds	Gets the seconds component of the time interval represented by the current TimeSpan structure.
Ticks	Gets the number of ticks that represent the value of the current TimeSpan structure.

TotalDays	Gets the value of the current TimeSpan structure expressed in whole and fractional days.
TotalHours	Gets the value of the current TimeSpan structure expressed in whole and fractional hours.
Total Microseconds	Gets the value of the current TimeSpan structure expressed in whole and fractional microseconds.
TotalMilliseconds	Gets the value of the current TimeSpan structure expressed in whole and fractional milliseconds.
TotalMinutes	Gets the value of the current TimeSpan structure expressed in whole and fractional minutes.
Total Nanoseconds	Gets the value of the current TimeSpan structure expressed in whole and fractional nanoseconds.
TotalSeconds	Gets the value of the current TimeSpan structure expressed in whole and fractional seconds.

Methods

Expand table

Add(TimeSpan)	Returns a new TimeSpan object whose value is the sum of the specified TimeSpan object and this instance.
Compare(TimeSpan, TimeSpan)	Compares two TimeSpan values and returns an integer that indicates whether the first value is shorter than, equal to, or longer than the second value.
CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the specified object.
CompareTo(TimeSpan)	Compares this instance to a specified TimeSpan object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the TimeSpan object.
Divide(Double)	Returns a new TimeSpan object whose value is the result of dividing this instance by the specified `divisor`.
Divide(TimeSpan)	Returns a new Double value that's the result of dividing this instance by `ts`.
Duration()	Returns a new TimeSpan object whose value is the absolute value of the current TimeSpan object.

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(TimeSpan)	Returns a value indicating whether this instance is equal to a specified TimeSpan object.
Equals(TimeSpan, TimeSpan)	Returns a value that indicates whether two specified instances of TimeSpan are equal.
FromDays(Double)	Returns a TimeSpan that represents a specified number of days, where the specification is accurate to the nearest millisecond.
FromHours(Double)	Returns a TimeSpan that represents a specified number of hours, where the specification is accurate to the nearest millisecond.
FromMicroseconds(Double)	Returns a TimeSpan that represents a specified number of microseconds.
FromMilliseconds(Double)	Returns a TimeSpan that represents a specified number of milliseconds.
FromMinutes(Double)	Returns a TimeSpan that represents a specified number of minutes, where the specification is accurate to the nearest millisecond.
FromSeconds(Double)	Returns a TimeSpan that represents a specified number of seconds, where the specification is accurate to the nearest millisecond.
FromTicks(Int64)	Returns a TimeSpan that represents a specified time, where the specification is in units of ticks.
GetHashCode()	Returns a hash code for this instance.
Multiply(Double)	Returns a new TimeSpan object which value is the result of multiplication of this instance and the specified `factor`.
Negate()	Returns a new TimeSpan object whose value is the negated value of this instance.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific format information.
Parse(String)	Converts the string representation of a time interval to its TimeSpan equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-

	specific format information.
ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles)	Converts the char span of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String, IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
ParseExact(String, String, IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles. The format of the string representation must match the specified format exactly.
ParseExact(String, String[], IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified array of format strings and culture-specific format information. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String[], IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.
Subtract(TimeSpan)	Returns a new TimeSpan object whose value is the difference between the specified TimeSpan object and this instance.
ToString()	Converts the value of the current TimeSpan object to its equivalent string representation.
ToString(String)	Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format.
ToString(String, IFormatProvider)	Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format and culture-specific formatting information.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current timespan number instance into the provided span of characters.
TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeSpan)	Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, TimeSpan)	Converts the span representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.
TryParse(String, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles, and returns a value that indicates whether the conversion succeeded. The format of the string representation must match the specified format exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.
TryParseExact(String, String, IFormatProvider, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and

	culture-specific format information. The format of the string representation must match the specified format exactly.
TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information and styles. The format of the string representation must match the specified format exactly.
TryParseExact(String, String[], IFormatProvider, TimeSpan)	Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.
TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.

Operators

Expand table

Addition(TimeSpan, TimeSpan)	Adds two specified TimeSpan instances.
Division(TimeSpan, Double)	Returns a new TimeSpan object whose value is the result of dividing the specified `timeSpan` by the specified `divisor`.
Division(TimeSpan, TimeSpan)	Returns a new Double value that's the result of dividing `t1` by `t2`.
Equality(TimeSpan, TimeSpan)	Indicates whether two TimeSpan instances are equal.
GreaterThan(TimeSpan, TimeSpan)	Indicates whether a specified TimeSpan is greater than another specified TimeSpan.
GreaterThanOrEqual(TimeSpan, TimeSpan)	Indicates whether a specified TimeSpan is greater than or equal to another specified TimeSpan.
Inequality(TimeSpan, TimeSpan)	Indicates whether two TimeSpan instances are not equal.
LessThan(TimeSpan, TimeSpan)	Indicates whether a specified TimeSpan is less than another specified TimeSpan.

LessThanOrEqual(TimeSpan, TimeSpan)	Indicates whether a specified TimeSpan is less than or equal to another specified TimeSpan.
Multiply(Double, TimeSpan)	Returns a new TimeSpan object whose value is the result of multiplying the specified `factor` and the specified `timeSpan` instance.
Multiply(TimeSpan, Double)	Returns a new TimeSpan object whose value is the result of multiplying the specified `timeSpan` instance and the specified `factor`.
Subtraction(TimeSpan, Time Span)	Subtracts a specified TimeSpan from another specified TimeSpan.
UnaryNegation(TimeSpan)	Returns a TimeSpan whose value is the negated value of the specified instance.
UnaryPlus(TimeSpan)	Returns the specified instance of TimeSpan.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- DateTime
- DateTimeOffset
- Calendar

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeSpan Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TimeSpan structure.

Overloads

Expand table

TimeSpan(Int64)	Initializes a new instance of the TimeSpan structure to the specified number of ticks.
TimeSpan(Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of hours, minutes, and seconds.
TimeSpan(Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, and seconds.
TimeSpan(Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, and milliseconds.
TimeSpan(Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, milliseconds, and microseconds.

TimeSpan(Int64)

Initializes a new instance of the TimeSpan structure to the specified number of ticks.

C#

```
public TimeSpan (long ticks);
```

Parameters

`ticks` Int64

A time period expressed in 100-nanosecond units.

Examples

The following example creates several TimeSpan objects using the constructor overload that initializes a TimeSpan to a specified number of ticks.

C#

```
// Example of the TimeSpan( long ) constructor.
using System;

class TimeSpanCtorLDemo
{
    // Create a TimeSpan object and display its value.
    static void CreateTimeSpan( long ticks )
    {
        TimeSpan elapsedTime = new TimeSpan( ticks );

        // Format the constructor for display.
        string ctor = String.Format( "TimeSpan( {0} )", ticks );

        // Pad the end of a TimeSpan string with spaces if
        // it does not contain milliseconds.
        string  elapsedStr = elapsedTime.ToString( );
        int     pointIndex = elapsedStr.IndexOf( ':' );

        pointIndex = elapsedStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) elapsedStr += "        ";

        // Display the constructor and its value.
        Console.WriteLine( "{0,-33}{1,24}", ctor, elapsedStr );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of the TimeSpan( long ) constructor " +
            "\ngenerates the following output.\n" );
        Console.WriteLine( "{0,-33}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-33}{1,16}", "-----------", "-----" );

        CreateTimeSpan( 1 );
        CreateTimeSpan( 999999 );
        CreateTimeSpan( -1000000000000 );
        CreateTimeSpan( 18012202000000 );
        CreateTimeSpan( 999999999999999999 );
        CreateTimeSpan( 1000000000000000000 );
    }
}

/*
```

```
This example of the TimeSpan( long ) constructor
generates the following output.

Constructor                                  Value
-----------                                  -----
TimeSpan( 1 )                              00:00:00.0000001
TimeSpan( 999999 )                         00:00:00.0999999
TimeSpan( -1000000000000 )             -1.03:46:40
TimeSpan( 18012202000000 )             20.20:20:20.2000000
TimeSpan( 999999999999999999 )    1157407.09:46:39.9999999
TimeSpan( 1000000000000000000 )   1157407.09:46:40
*/
```

Remarks

A single tick represents one hundred nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond.

See also

- Ticks

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeSpan(Int32, Int32, Int32)

Initializes a new instance of the TimeSpan structure to a specified number of hours, minutes, and seconds.

```
C#
```

```
public TimeSpan (int hours, int minutes, int seconds);
```

Parameters

hours Int32

Number of hours.

minutes Int32

Number of minutes.

seconds Int32

Number of seconds.

Exceptions

ArgumentOutOfRangeException

The parameters specify a TimeSpan value less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example creates several TimeSpan objects using the constructor overload that initializes a TimeSpan to a specified number of hours, minutes, and seconds.

C#

```
// Example of the TimeSpan( int, int, int ) constructor.
using System;

class TimeSpanCtorIIIDemo
{
    // Create a TimeSpan object and display its value.
    static void CreateTimeSpan( int hours, int minutes,
        int seconds )
    {
        TimeSpan elapsedTime =
            new TimeSpan( hours, minutes, seconds );

        // Format the constructor for display.
        string ctor = String.Format( "TimeSpan( {0}, {1}, {2} )",
            hours, minutes, seconds);

        // Display the constructor and its value.
        Console.WriteLine( "{0,-37}{1,16}",
```

```
            ctor, elapsedTime.ToString( ) );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of the TimeSpan( int, int, int ) " +
            "\nconstructor generates the following output.\n" );
        Console.WriteLine( "{0,-37}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-37}{1,16}", "-----------", "-----" );

        CreateTimeSpan( 10, 20, 30 );
        CreateTimeSpan( -10, 20, 30 );
        CreateTimeSpan( 0, 0, 37230 );
        CreateTimeSpan( 1000, 2000, 3000 );
        CreateTimeSpan( 1000, -2000, -3000 );
        CreateTimeSpan( 999999, 999999, 999999 );
    }
}

/*
This example of the TimeSpan( int, int, int )
constructor generates the following output.

Constructor                                     Value
-----------                                     -----
TimeSpan( 10, 20, 30 )                       10:20:30
TimeSpan( -10, 20, 30 )                     -09:39:30
TimeSpan( 0, 0, 37230 )                      10:20:30
TimeSpan( 1000, 2000, 3000 )              43.02:10:00
TimeSpan( 1000, -2000, -3000 )            40.05:50:00
TimeSpan( 999999, 999999, 999999 )     42372.15:25:39
*/
```

Remarks

The specified `hours`, `minutes`, and `seconds` are converted to ticks, and that value initializes this instance.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeSpan(Int32, Int32, Int32, Int32)

Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, and seconds.

C#

```
public TimeSpan (int days, int hours, int minutes, int seconds);
```

Parameters

`days` Int32

Number of days.

`hours` Int32

Number of hours.

`minutes` Int32

Number of minutes.

`seconds` Int32

Number of seconds.

Exceptions

ArgumentOutOfRangeException

The parameters specify a TimeSpan value less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example creates several TimeSpan objects using the constructor overload that initializes a TimeSpan to a specified number of days, hours, minutes, and seconds.

C#

```
using System;

class Example
{
    // Create a TimeSpan object and display its value.
    static void CreateTimeSpan( int days, int hours,
        int minutes, int seconds )
    {
        TimeSpan elapsedTime =
            new TimeSpan( days, hours, minutes, seconds );

        // Format the constructor for display.
        string ctor =
            String.Format( "TimeSpan( {0}, {1}, {2}, {3} )",
                days, hours, minutes, seconds);

        // Display the constructor and its value.
        Console.WriteLine( "{0,-44}{1,16}",
            ctor, elapsedTime.ToString( ) );
    }

    static void Main( )
    {
        Console.WriteLine( "{0,-44}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-44}{1,16}", "-----------", "-----" );

        CreateTimeSpan( 10, 20, 30, 40 );
        CreateTimeSpan( -10, 20, 30, 40 );
        CreateTimeSpan( 0, 0, 0, 937840 );
        CreateTimeSpan( 1000, 2000, 3000, 4000 );
        CreateTimeSpan( 1000, -2000, -3000, -4000 );
        CreateTimeSpan( 999999, 999999, 999999, 999999 );
    }
}
// The example displays the following output:
//       Constructor                                          Value
//       -----------                                          -----
//       TimeSpan( 10, 20, 30, 40 )                      10.20:30:40
//       TimeSpan( -10, 20, 30, 40 )                     -9.03:29:20
//       TimeSpan( 0, 0, 0, 937840 )                     10.20:30:40
//       TimeSpan( 1000, 2000, 3000, 4000 )            1085.11:06:40
//       TimeSpan( 1000, -2000, -3000, -4000 )          914.12:53:20
//       TimeSpan( 999999, 999999, 999999, 999999 )  1042371.15:25:39
```

Remarks

The specified `days`, `hours`, `minutes`, and `seconds` are converted to ticks, and that value initializes this instance.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeSpan(Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, and milliseconds.

C#

```
public TimeSpan (int days, int hours, int minutes, int seconds, int
milliseconds);
```

Parameters

days Int32

Number of days.

hours Int32

Number of hours.

`minutes` Int32

Number of minutes.

`seconds` Int32

Number of seconds.

`milliseconds` Int32

Number of milliseconds.

Exceptions

ArgumentOutOfRangeException

The parameters specify a TimeSpan value less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example creates several TimeSpan objects using the constructor overload that initializes a TimeSpan to a specified number of days, hours, minutes, seconds, and milliseconds.

C#

```
// Example of the TimeSpan( int, int, int, int, int ) constructor.
using System;

class TimeSpanCtorIIIIIDemo
{
    // Create a TimeSpan object and display its value.
    static void CreateTimeSpan( int days, int hours,
        int minutes, int seconds, int millisec )
    {
        TimeSpan elapsedTime = new TimeSpan(
            days, hours, minutes, seconds, millisec );

        // Format the constructor for display.
        string ctor =
            String.Format( "TimeSpan( {0}, {1}, {2}, {3}, {4} )",
                days, hours, minutes, seconds, millisec);

        // Display the constructor and its value.
        Console.WriteLine( "{0,-48}{1,24}",
            ctor, elapsedTime.ToString( ) );
    }

    static void Main( )
    {
```

```
        Console.WriteLine(
            "This example of the " +
            "TimeSpan( int, int, int, int, int ) " +
            "\nconstructor generates the following output.\n" );
        Console.WriteLine( "{0,-48}{1,16}", "Constructor", "Value" );
        Console.WriteLine( "{0,-48}{1,16}", "-----------", "-----" );

        CreateTimeSpan( 10, 20, 30, 40, 50 );
        CreateTimeSpan( -10, 20, 30, 40, 50 );
        CreateTimeSpan( 0, 0, 0, 0, 937840050 );
        CreateTimeSpan( 1111, 2222, 3333, 4444, 5555 );
        CreateTimeSpan( 1111, -2222, -3333, -4444, -5555 );
        CreateTimeSpan( 99999, 99999, 99999, 99999, 99999 );
    }
}

/*
This example of the TimeSpan( int, int, int, int, int )
constructor generates the following output.

Constructor                                                Value
-----------                                                -----
TimeSpan( 10, 20, 30, 40, 50 )                       10.20:30:40.0500000
TimeSpan( -10, 20, 30, 40, 50 )                      -9.03:29:19.9500000
TimeSpan( 0, 0, 0, 0, 937840050 )                    10.20:30:40.0500000
TimeSpan( 1111, 2222, 3333, 4444, 5555 )           1205.22:47:09.5550000
TimeSpan( 1111, -2222, -3333, -4444, -5555 )       1016.01:12:50.4450000
TimeSpan( 99999, 99999, 99999, 99999, 99999 )    104236.05:27:18.9990000
*/
```

Remarks

The specified `days`, `hours`, `minutes`, `seconds`, and `milliseconds` are converted to ticks, and that value initializes this instance.

See also

- Int64

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TimeSpan(Int32, Int32, Int32, Int32, Int32, Int32)

Initializes a new instance of the TimeSpan structure to a specified number of days, hours, minutes, seconds, milliseconds, and microseconds.

C#

```
public TimeSpan (int days, int hours, int minutes, int seconds, int
milliseconds, int microseconds);
```

Parameters

days Int32

Number of days.

hours Int32

Number of hours.

minutes Int32

Number of minutes.

seconds Int32

Number of seconds.

milliseconds Int32

Number of milliseconds.

microseconds Int32

Number of microseconds.

Exceptions

ArgumentOutOfRangeException

The parameters specify a TimeSpan value less than MinValue or greater than MaxValue

Remarks

The specified `days`, `hours`, `minutes`, `seconds`, `milliseconds`, and `microseconds` are converted to ticks, and that value initializes this instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the maximum TimeSpan value. This field is read-only.

C#

```
public static readonly TimeSpan MaxValue;
```

Field Value

TimeSpan

Examples

The following example references and displays the value of the MaxValue field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
        const string timeFmt = "{0,-22}{1,26}" ;
```

```
        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                              Value
-----                              -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this field is equivalent to Int64.MaxValue ticks. The string representation of this value is positive 10675199.02:48:05.4775807, or slightly more than 10,675,199 days.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the minimum TimeSpan value. This field is read-only.

C#

```
public static readonly TimeSpan MinValue;
```

Field Value

TimeSpan

Examples

The following example references and displays the value of the MinValue field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
        const string timeFmt = "{0,-22}{1,26}" ;
```

```
        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                              Value
-----                              -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this field is equivalent to Int64.MinValue ticks. The string representation of this value is negative 10675199.02:48:05.4775808, or slightly more than negative

10,675,199 days.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.NanosecondsPerTick Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of nanoseconds per tick. This field is constant.

C#

```
public const long NanosecondsPerTick = 100;
```

Field Value

Value* = *100

Int64

Remarks

The value of this constant is 100.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.TicksPerDay Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 day. This field is constant.

C#

```
public const long TicksPerDay = 864000000000;
```

Field Value

Value* = *864000000000

Int64

Examples

The following example references and displays the value of the TicksPerDay field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
```

```
        const string timeFmt = "{0,-22}{1,26}" ;

        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                                Value
-----                                -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this constant is 864 billion; that is, 864,000,000,000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- TicksPerMillisecond
- TicksPerSecond
- TicksPerMinute
- TicksPerHour

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TicksPerHour Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 hour. This field is constant.

C#

```
public const long TicksPerHour = 36000000000;
```

Field Value

***Value* = 36000000000**

Int64

Examples

The following example references and displays the value of the TicksPerHour field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
```

```
        const string timeFmt = "{0,-22}{1,26}" ;

        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                                Value
-----                                -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this constant is 36 billion; that is, 36,000,000,000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- TicksPerMillisecond
- TicksPerSecond
- TicksPerMinute
- TicksPerDay

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TicksPerMicrosecond Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 microsecond. This field is constant.

C#

```
public const long TicksPerMicrosecond = 10;
```

Field Value

Value* = *10

Int64

Remarks

The value of this constant is 10.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.TicksPerMillisecond Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 millisecond. This field is constant.

C#

```
public const long TicksPerMillisecond = 10000;
```

Field Value

Value* = *10000

Int64

Examples

The following example references and displays the value of the TicksPerMillisecond field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
```

```
        const string timeFmt = "{0,-22}{1,26}" ;

        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                                Value
-----                                -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this constant is 10 thousand; that is, 10,000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- TicksPerSecond
- TicksPerMinute
- TicksPerHour
- TicksPerDay

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TicksPerMinute Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 minute. This field is constant.

C#

```
public const long TicksPerMinute = 600000000;
```

Field Value

***Value* = 600000000**

Int64

Examples

The following example references and displays the value of the TicksPerMinute field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
```

```
        const string timeFmt = "{0,-22}{1,26}" ;

        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                                Value
-----                                -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this constant is 600 million; that is, 600,000,000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- TicksPerMillisecond
- TicksPerSecond
- TicksPerHour
- TicksPerDay

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TicksPerSecond Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the number of ticks in 1 second.

C#

```
public const long TicksPerSecond = 10000000;
```

Field Value

Value* = *10000000

Int64

Examples

The following example references and displays the value of the TicksPerSecond field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
```

```
        const string timeFmt = "{0,-22}{1,26}" ;

        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                                Value
-----                                -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

The value of this constant is 10 million; that is, 10,000,000.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- TicksPerMillisecond
- TicksPerMinute
- TicksPerHour
- TicksPerDay

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Zero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the zero TimeSpan value. This field is read-only.

C#

```
public static readonly TimeSpan Zero;
```

Field Value

TimeSpan

Examples

The following example references and displays the value of the Zero field.

C#

```
// Example of the TimeSpan fields.
using System;

class TimeSpanFieldsDemo
{
    // Pad the end of a TimeSpan string with spaces if it does not
    // contain milliseconds.
    static string Align( TimeSpan interval )
    {
        string  intervalStr = interval.ToString( );
        int     pointIndex = intervalStr.IndexOf( ':' );

        pointIndex = intervalStr.IndexOf( '.', pointIndex );
        if( pointIndex < 0 ) intervalStr += "        ";
        return intervalStr;
    }

    static void Main( )
    {
        const string numberFmt = "{0,-22}{1,18:N0}" ;
        const string timeFmt = "{0,-22}{1,26}" ;
```

```
        Console.WriteLine(
            "This example of the fields of the TimeSpan class" +
            "\ngenerates the following output.\n" );
        Console.WriteLine( numberFmt, "Field", "Value" );
        Console.WriteLine( numberFmt, "-----", "-----" );

        // Display the maximum, minimum, and zero TimeSpan values.
        Console.WriteLine( timeFmt, "Maximum TimeSpan",
            Align( TimeSpan.MaxValue ) );
        Console.WriteLine( timeFmt, "Minimum TimeSpan",
            Align( TimeSpan.MinValue ) );
        Console.WriteLine( timeFmt, "Zero TimeSpan",
            Align( TimeSpan.Zero ) );
        Console.WriteLine( );

        // Display the ticks-per-time-unit fields.
        Console.WriteLine( numberFmt, "Ticks per day",
            TimeSpan.TicksPerDay );
        Console.WriteLine( numberFmt, "Ticks per hour",
            TimeSpan.TicksPerHour );
        Console.WriteLine( numberFmt, "Ticks per minute",
            TimeSpan.TicksPerMinute );
        Console.WriteLine( numberFmt, "Ticks per second",
            TimeSpan.TicksPerSecond );
        Console.WriteLine( numberFmt, "Ticks per millisecond",
            TimeSpan.TicksPerMillisecond );
    }
}

/*
This example of the fields of the TimeSpan class
generates the following output.

Field                              Value
-----                              -----
Maximum TimeSpan       10675199.02:48:05.4775807
Minimum TimeSpan      -10675199.02:48:05.4775808
Zero TimeSpan                   00:00:00

Ticks per day            864,000,000,000
Ticks per hour            36,000,000,000
Ticks per minute             600,000,000
Ticks per second              10,000,000
Ticks per millisecond             10,000
*/
```

Remarks

Because the value of the Zero field is a TimeSpan object that represents a zero time value, you can compare it with other TimeSpan objects to determine whether the latter

represent positive, non-zero, or negative time intervals. You can also use this field to initialize a TimeSpan object to a zero time value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Days Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the days component of the time interval represented by the current TimeSpan structure.

C#

```
public int Days { get; }
```

Property Value

Int32

The day component of this instance. The return value can be positive or negative.

Examples

The following example creates several TimeSpan objects and displays the Days property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
```

```
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));

        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                     00:00:00.0000001
//    Days               0       TotalDays                         0.000
//    Hours              0       TotalHours                        0.000
//    Minutes            0       TotalMinutes                      0.000
//    Seconds            0       TotalSeconds                      0.000
//    Milliseconds       0       TotalMilliseconds                 0.000
//                               Ticks                                 1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                       128.730
//    Hours             17       TotalHours                    3,089.509
//    Minutes           30       TotalMinutes                185,370.556
//    Seconds           33       TotalSeconds              11,122,233.344
//    Milliseconds     344       TotalMilliseconds     11,122,233,344.456
//                               Ticks                111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                  10.20:30:40.0500000
//    Days              10       TotalDays                        10.855
```

```
//    Hours             20       TotalHours                     260.511
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

A TimeSpan value can be represented as [-]*d.hh*:*mm*:*ss.ff*, where the optional minus sign indicates a negative time interval, the *d* component is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second. The value of the Days property is the day component, *d*.

The Days property represents whole days, whereas the TotalDays property represents whole and fractional days.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TotalDays
- Ticks
- Milliseconds
- Seconds
- Minutes
- Hours

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Hours Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hours component of the time interval represented by the current TimeSpan structure.

C#

```
public int Hours { get; }
```

Property Value

Int32

The hour component of the current TimeSpan structure. The return value ranges from -23 through 23.

Examples

The following example creates several TimeSpan objects and displays the Hours property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
```

```
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));

        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                        00:00:00.0000001
//    Days               0       TotalDays                          0.000
//    Hours              0       TotalHours                         0.000
//    Minutes            0       TotalMinutes                       0.000
//    Seconds            0       TotalSeconds                       0.000
//    Milliseconds       0       TotalMilliseconds                  0.000
//                               Ticks                                  1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                        128.730
//    Hours             17       TotalHours                     3,089.509
//    Minutes           30       TotalMinutes                 185,370.556
//    Seconds           33       TotalSeconds               11,122,233.344
//    Milliseconds     344       TotalMilliseconds      11,122,233,344.456
//                               Ticks                 111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                  10.20:30:40.0500000
```

```
//    Days              10       TotalDays                       10.855
//    Hours             20       TotalHours                     260.511
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

A TimeSpan value can be represented as [-]*d.hh*:*mm*:*ss.ff*, where the optional minus sign indicates a negative time interval, the *d* component is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second. The value of the Hours property is the hours component, *hh*.

The Hours property represents whole hours, whereas the TotalHours property represents whole and fractional hours.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TotalHours
- Ticks
- Milliseconds
- Seconds
- Minutes
- Days

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Microseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the microseconds component of the time interval represented by the current TimeSpan structure.

C#

```
public int Microseconds { get; }
```

Property Value

Int32

Remarks

The Microseconds property represents whole microseconds, whereas the TotalMicroseconds property represents whole and fractional microseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.Milliseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the milliseconds component of the time interval represented by the current TimeSpan structure.

C#

```
public int Milliseconds { get; }
```

Property Value

Int32

The millisecond component of the current TimeSpan structure. The return value ranges from -999 through 999.

Examples

The following example creates several TimeSpan objects and displays the Milliseconds property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
```

```
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));

        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                        00:00:00.0000001
//    Days               0       TotalDays                          0.000
//    Hours              0       TotalHours                         0.000
//    Minutes            0       TotalMinutes                       0.000
//    Seconds            0       TotalSeconds                       0.000
//    Milliseconds       0       TotalMilliseconds                  0.000
//                               Ticks                                  1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                        128.730
//    Hours             17       TotalHours                     3,089.509
//    Minutes           30       TotalMinutes                 185,370.556
//    Seconds           33       TotalSeconds               11,122,233.344
//    Milliseconds     344       TotalMilliseconds      11,122,233,344.456
//                               Ticks                 111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                  10.20:30:40.0500000
```

```
//    Days              10       TotalDays                       10.855
//    Hours             20       TotalHours                     260.511
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

A TimeSpan value can be represented as [-]*d.hh*:*mm*:*ss.ff*, where the optional minus sign indicates a negative time interval, the *d* component is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second. The value of the Milliseconds property is the fractional second component, *ff*.

The Milliseconds property represents whole milliseconds, whereas the TotalMilliseconds property represents whole and fractional milliseconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TotalMilliseconds
- Ticks
- Seconds
- Minutes
- Hours
- Days

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Minutes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minutes component of the time interval represented by the current TimeSpan structure.

C#

```
public int Minutes { get; }
```

Property Value

Int32

The minute component of the current TimeSpan structure. The return value ranges from -59 through 59.

Examples

The following example creates several TimeSpan objects and displays the Minutes property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
```

```
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));

        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                        00:00:00.0000001
//    Days               0       TotalDays                          0.000
//    Hours              0       TotalHours                         0.000
//    Minutes            0       TotalMinutes                       0.000
//    Seconds            0       TotalSeconds                       0.000
//    Milliseconds       0       TotalMilliseconds                  0.000
//                               Ticks                                  1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                        128.730
//    Hours             17       TotalHours                     3,089.509
//    Minutes           30       TotalMinutes                 185,370.556
//    Seconds           33       TotalSeconds               11,122,233.344
//    Milliseconds     344       TotalMilliseconds      11,122,233,344.456
//                               Ticks                 111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                  10.20:30:40.0500000
```

```
//    Days              10       TotalDays                       10.855
//    Hours             20       TotalHours                     260.511
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

A TimeSpan value can be represented as [-]*d.hh*:*mm*:*ss.ff*, where the optional minus sign indicates a negative time interval, the *d* component is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second. The value of the Minutes property is the minute component, *mm*.

The Minutes property represents whole minutes, whereas the TotalMinutes property represents whole and fractional minutes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TotalMinutes
- Ticks
- Milliseconds
- Seconds
- Hours
- Days

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Nanoseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the nanoseconds component of the time interval represented by the current TimeSpan structure.

C#

```
public int Nanoseconds { get; }
```

Property Value

Int32

Remarks

The Nanoseconds property represents whole nanoseconds, whereas the TotalNanoseconds property represents whole and fractional nanoseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.Seconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the seconds component of the time interval represented by the current TimeSpan structure.

C#

```
public int Seconds { get; }
```

Property Value

Int32

The second component of the current TimeSpan structure. The return value ranges from -59 through 59.

Examples

The following example creates several TimeSpan objects and displays the Seconds property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
```

```
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));

        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                        00:00:00.0000001
//    Days               0       TotalDays                          0.000
//    Hours              0       TotalHours                         0.000
//    Minutes            0       TotalMinutes                       0.000
//    Seconds            0       TotalSeconds                       0.000
//    Milliseconds       0       TotalMilliseconds                  0.000
//                               Ticks                                  1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                        128.730
//    Hours             17       TotalHours                     3,089.509
//    Minutes           30       TotalMinutes                 185,370.556
//    Seconds           33       TotalSeconds               11,122,233.344
//    Milliseconds     344       TotalMilliseconds      11,122,233,344.456
//                               Ticks                 111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                  10.20:30:40.0500000
```

```
//    Days              10       TotalDays                       10.855
//    Hours             20       TotalHours                     260.511
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

A TimeSpan value can be represented as [-]*d.hh*:*mm*:*ss.ff*, where the optional minus sign indicates a negative time interval, the *d* component is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second. The value of the Seconds property is the seconds component, *ss*.

The Seconds property represents whole seconds, whereas the TotalSeconds property represents whole and fractional seconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- TotalSeconds
- Ticks
- Milliseconds
- Minutes
- Hours
- Days

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Ticks Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of ticks that represent the value of the current TimeSpan structure.

C#

```
public long Ticks { get; }
```

Property Value

Int64

The number of ticks contained in this instance.

Examples

The following example creates several TimeSpan objects and displays the Ticks property of each.

C#

```
using System;

class Example
{
    static void Main()
    {
        // Create and display a TimeSpan value of 1 tick.
        Console.Write("\n{0,-45}", "TimeSpan( 1 )");
        ShowTimeSpanProperties(new TimeSpan(1));

        // Create a TimeSpan value with a large number of ticks.
        Console.Write("\n{0,-45}", "TimeSpan( 111222333444555 )");
        ShowTimeSpanProperties(new TimeSpan(111222333444555));

        // This TimeSpan has all fields specified.
        Console.Write("\n{0,-45}", "TimeSpan( 10, 20, 30, 40, 50 )");
        ShowTimeSpanProperties(new TimeSpan(10, 20, 30, 40, 50));
```

```
        // This TimeSpan has all fields overflowing.
        Console.Write("\n{0,-45}",
                      "TimeSpan( 1111, 2222, 3333, 4444, 5555 )");
        ShowTimeSpanProperties(
           new TimeSpan(1111, 2222, 3333, 4444, 5555));

        // This TimeSpan is based on a number of days.
        Console.Write("\n{0,-45}", "FromDays( 20.84745602 )");
        ShowTimeSpanProperties(TimeSpan.FromDays( 20.84745602));
    }

    static void ShowTimeSpanProperties( TimeSpan interval )
    {
        Console.WriteLine("{0,21}", interval);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Days",
                          interval.Days, "TotalDays", interval.TotalDays);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Hours",
                          interval.Hours, "TotalHours",
interval.TotalHours);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Minutes",
                          interval.Minutes, "TotalMinutes",
interval.TotalMinutes);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}", "Seconds",
                          interval.Seconds, "TotalSeconds",
interval.TotalSeconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N3}",
"Milliseconds",
                          interval.Milliseconds, "TotalMilliseconds",
                          interval.TotalMilliseconds);
        Console.WriteLine("{0,-12}{1,8}       {2,-18}{3,21:N0}", null, null,
            "Ticks", interval.Ticks);
    }
}
// The example displays the following output if the current culture is en-
US:
//    TimeSpan( 1 )                                       00:00:00.0000001
//    Days               0       TotalDays                          0.000
//    Hours              0       TotalHours                         0.000
//    Minutes            0       TotalMinutes                       0.000
//    Seconds            0       TotalSeconds                       0.000
//    Milliseconds       0       TotalMilliseconds                  0.000
//                               Ticks                                  1
//
//    TimeSpan( 111222333444555 )                   128.17:30:33.3444555
//    Days             128       TotalDays                        128.730
//    Hours             17       TotalHours                     3,089.509
//    Minutes           30       TotalMinutes                 185,370.556
//    Seconds           33       TotalSeconds               11,122,233.344
//    Milliseconds     344       TotalMilliseconds      11,122,233,344.456
//                               Ticks                 111,222,333,444,555
//
//    TimeSpan( 10, 20, 30, 40, 50 )                 10.20:30:40.0500000
//    Days              10       TotalDays                         10.855
//    Hours             20       TotalHours                       260.511
```

```
//    Minutes           30       TotalMinutes                15,630.668
//    Seconds           40       TotalSeconds               937,840.050
//    Milliseconds      50       TotalMilliseconds       937,840,050.000
//                               Ticks                 9,378,400,500,000
//
//    TimeSpan( 1111, 2222, 3333, 4444, 5555 )     1205.22:47:09.5550000
//    Days            1205       TotalDays                    1,205.949
//    Hours             22       TotalHours                  28,942.786
//    Minutes           47       TotalMinutes             1,736,567.159
//    Seconds            9       TotalSeconds           104,194,029.555
//    Milliseconds     555       TotalMilliseconds   104,194,029,555.000
//                               Ticks             1,041,940,295,550,000
//
//    FromDays( 20.84745602 )                          20.20:20:20.2000000
//    Days              20       TotalDays                       20.847
//    Hours             20       TotalHours                     500.339
//    Minutes           20       TotalMinutes                30,020.337
//    Seconds           20       TotalSeconds             1,801,220.200
//    Milliseconds     200       TotalMilliseconds     1,801,220,200.000
//                               Ticks                18,012,202,000,000
```

Remarks

The smallest unit of time is the tick, which is equal to 100 nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond. The value of the Ticks property can be negative or positive to represent a negative or positive time interval.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- Milliseconds
- Seconds
- Minutes
- Hours

- Days

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TotalDays Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional days.

C#

```
public double TotalDays { get; }
```

Property Value

Double

The total number of days represented by this instance.

Examples

The following example instantiates a TimeSpan object and displays the value of its TotalDays property. It also displays the value of each component (hours, minutes, seconds, milliseconds) that forms the fractional part of the value of its TotalDays property.

C#

```
// Define an interval of 3 days, 16+ hours.
TimeSpan interval = new TimeSpan(3, 16, 42, 45, 750);
Console.WriteLine("Value of TimeSpan: {0}", interval);

Console.WriteLine("{0:N5} days, as follows:", interval.TotalDays);
Console.WriteLine("   Days:         {0,3}", interval.Days);
Console.WriteLine("   Hours:        {0,3}", interval.Hours);
Console.WriteLine("   Minutes:      {0,3}", interval.Minutes);
Console.WriteLine("   Seconds:      {0,3}", interval.Seconds);
Console.WriteLine("   Milliseconds: {0,3}", interval.Milliseconds);

// The example displays the following output:
//       Value of TimeSpan: 3.16:42:45.7500000
//       3.69636 days, as follows:
//          Days:           3
```

```
//          Hours:         16
//          Minutes:       42
//          Seconds:       45
//          Milliseconds: 750
```

Remarks

This property converts the value of this instance from ticks to days. This number might include whole and fractional days.

The TotalDays property represents whole and fractional days, whereas the Days property represents whole days.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Days
- TotalMilliseconds
- TotalSeconds
- TotalMinutes
- TotalHours

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.TotalHours Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional hours.

C#

```
public double TotalHours { get; }
```

Property Value

Double

The total number of hours represented by this instance.

Examples

The following example instantiates a TimeSpan object and displays the value its TotalHours property. It also displays the value of each component (hours, minutes, seconds, and milliseconds) that forms the fractional part of the value of its TotalHours property.

C#

```
// Define an interval of 1 day, 15+ hours.
TimeSpan interval = new TimeSpan(1, 15, 42, 45, 750);
Console.WriteLine("Value of TimeSpan: {0}", interval);

Console.WriteLine("{0:N5} hours, as follows:", interval.TotalHours);
Console.WriteLine("   Hours:        {0,3}",
                  interval.Days * 24 + interval.Hours);
Console.WriteLine("   Minutes:      {0,3}", interval.Minutes);
Console.WriteLine("   Seconds:      {0,3}", interval.Seconds);
Console.WriteLine("   Milliseconds: {0,3}", interval.Milliseconds);

// The example displays the following output:
//       Value of TimeSpan: 1.15:42:45.7500000
```

```
//        39.71271 hours, as follows:
//           Hours:         39
//           Minutes:       42
//           Seconds:       45
//           Milliseconds: 750
```

Remarks

This property converts the value of this instance from ticks to hours. This number might include whole and fractional hours.

The TotalHours property represents whole and fractional hours, whereas the Hours property represents whole hours.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Hours
- TotalMilliseconds
- TotalSeconds
- TotalMinutes
- TotalDays

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeSpan.TotalMicroseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional microseconds.

C#

```
public double TotalMicroseconds { get; }
```

Property Value

Double

Remarks

This property converts the value of this instance from ticks to microseconds. This number might include whole and fractional microseconds.

The TotalMicroseconds property represents whole and fractional microseconds, whereas the Microseconds property represents whole microseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeSpan.TotalMilliseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional milliseconds.

C#

```
public double TotalMilliseconds { get; }
```

Property Value

Double

The total number of milliseconds represented by this instance.

Examples

The following example instantiates a TimeSpan object and displays the value of its TotalMilliseconds property.

C#

```
// Define an interval of 1 day, 15+ hours.
TimeSpan interval = new TimeSpan(1, 15, 42, 45, 750);
Console.WriteLine("Value of TimeSpan: {0}", interval);

Console.WriteLine("There are {0:N5} milliseconds, as follows:",
interval.TotalMilliseconds);
long nMilliseconds = interval.Days * 24 * 60 * 60 * 1000 +
                     interval.Hours *60 * 60 * 1000 +
                     interval.Minutes * 60 * 1000 +
                     interval.Seconds * 1000 +
                     interval.Milliseconds;
Console.WriteLine("   Milliseconds:     {0,18:N0}", nMilliseconds);
Console.WriteLine("   Ticks:            {0,18:N0}",
                  nMilliseconds * 10000 - interval.Ticks);

// The example displays the following output:
```

```
//       Value of TimeSpan: 1.15:42:45.7500000
//       There are 142,965,750.00000 milliseconds, as follows:
//          Milliseconds:            142,965,750
//          Ticks:                             0
```

Remarks

This property converts the value of this instance from ticks to milliseconds. This number might include whole and fractional milliseconds.

The TotalMilliseconds property represents whole and fractional milliseconds, whereas the Milliseconds property represents whole milliseconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Milliseconds
- TotalSeconds
- TotalMinutes
- TotalHours
- TotalDays

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.TotalMinutes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional minutes.

C#

```
public double TotalMinutes { get; }
```

Property Value

Double

The total number of minutes represented by this instance.

Examples

The following example instantiates a TimeSpan object and displays the value of its TotalMinutes property. It also displays the value of each component (seconds, milliseconds) that forms the fractional part of the value of its TotalMinutes property.

C#

```
// Define an interval of 1 day, 15+ hours.
TimeSpan interval = new TimeSpan(1, 15, 42, 45, 750);
Console.WriteLine("Value of TimeSpan: {0}", interval);

Console.WriteLine("{0:N5} minutes, as follows:", interval.TotalMinutes);
Console.WriteLine("   Minutes:      {0,5}", interval.Days * 24 * 60 +
                                             interval.Hours * 60 +
                                             interval.Minutes);
Console.WriteLine("   Seconds:      {0,5}", interval.Seconds);
Console.WriteLine("   Milliseconds: {0,5}", interval.Milliseconds);

// The example displays the following output:
//       Value of TimeSpan: 1.15:42:45.7500000
//       2,382.76250 minutes, as follows:
//          Minutes:       2382
```

```
//          Seconds:         45
//          Milliseconds:   750
```

Remarks

This property converts the value of this instance from ticks to minutes. This number might include whole and fractional minutes.

The TotalMinutes property represents whole and fractional minutes, whereas the Minutes property represents whole minutes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Minutes
- TotalMilliseconds
- TotalSeconds
- TotalHours
- TotalDays

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TotalNanoseconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional nanoseconds.

C#

```
public double TotalNanoseconds { get; }
```

Property Value

Double

Remarks

This property converts the value of this instance from ticks to nanoseconds. This number might include whole and fractional nanoseconds.

The TotalNanoseconds property represents whole and fractional nanoseconds, whereas the Nanoseconds property represents whole nanoseconds.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeSpan.TotalSeconds Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current TimeSpan structure expressed in whole and fractional seconds.

C#

```
public double TotalSeconds { get; }
```

Property Value

Double

The total number of seconds represented by this instance.

Examples

The following example instantiates a TimeSpan object and displays the value of its TotalSeconds property. It also displays the value of its milliseconds component, which forms the fractional part of the value of its TotalSeconds property.

C#

```
// Define an interval of 1 day, 15+ hours.
TimeSpan interval = new TimeSpan(1, 15, 42, 45, 750);
Console.WriteLine("Value of TimeSpan: {0}", interval);

Console.WriteLine("{0:N5} seconds, as follows:", interval.TotalSeconds);
Console.WriteLine("   Seconds:      {0,8:N0}", interval.Days * 24 * 60 * 60
+
                                                interval.Hours *60 * 60 +
                                                interval.Minutes * 60 +
                                                interval.Seconds);
Console.WriteLine("   Milliseconds: {0,8}", interval.Milliseconds);

// The example displays the following output:
//       Value of TimeSpan: 1.15:42:45.7500000
//       142,965.75000 seconds, as follows:
```

```
//          Seconds:       142,965
//          Milliseconds:      750
```

Remarks

This property converts the value of this instance from ticks to seconds. This number might include whole and fractional seconds.

The TotalSeconds property represents whole and fractional seconds, whereas the Seconds property represents whole seconds.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Seconds
- TotalMilliseconds
- TotalMinutes
- TotalHours
- TotalDays

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Add(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new TimeSpan object whose value is the sum of the specified TimeSpan object and this instance.

C#

```
public TimeSpan Add (TimeSpan ts);
```

Parameters

`ts` TimeSpan

The time interval to add.

Returns

TimeSpan

A new object that represents the value of this instance plus the value of `ts`.

Exceptions

OverflowException

The resulting TimeSpan is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example calls the Add method to add each element in an array of time intervals to a base TimeSpan value.

C#

```
TimeSpan baseTimeSpan = new TimeSpan(1, 12, 15, 16);
```

```
// Create an array of timespan intervals.
TimeSpan[] intervals = {
   TimeSpan.FromDays(1.5),
   TimeSpan.FromHours(1.5),
   TimeSpan.FromMinutes(45),
   TimeSpan.FromMilliseconds(505),
   new TimeSpan(1, 17, 32, 20),
   new TimeSpan(-8, 30, 0)
};

// Calculate a new time interval by adding each element to the base
interval.
foreach (var interval in intervals)
   Console.WriteLine(@"{0,-10:g} {3} {1,15:%d\:hh\:mm\:ss\.ffff} =
{2:%d\:hh\:mm\:ss\.ffff}",
                     baseTimeSpan, interval, baseTimeSpan.Add(interval),
                     interval < TimeSpan.Zero ? "-" : "+");

// The example displays the following output:
//       1:12:15:16 + 1:12:00:00.0000 = 3:00:15:16.0000
//       1:12:15:16 + 0:01:30:00.0000 = 1:13:45:16.0000
//       1:12:15:16 + 0:00:45:00.0000 = 1:13:00:16.0000
//       1:12:15:16 + 0:00:00:00.5050 = 1:12:15:16.5050
//       1:12:15:16 + 1:17:32:20.0000 = 3:05:47:36.0000
//       1:12:15:16 - 0:07:30:00.0000 = 1:04:45:16.0000
```

Remarks

The return value must be between TimeSpan.MinValue and TimeSpan.MaxValue; otherwise, an exception is thrown.

The return value is a new TimeSpan; the original TimeSpan is not modified.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(TimeSpan)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Compare(TimeSpan, TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two TimeSpan values and returns an integer that indicates whether the first value is shorter than, equal to, or longer than the second value.

C#

```
public static int Compare (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Int32

One of the following values.

Expand table

Value	Description
-1	`t1` is shorter than `t2`.
0	`t1` is equal to `t2`.
1	`t1` is longer than `t2`.

Examples

The following example uses the Compare method to compare several TimeSpan objects with a TimeSpan object whose value is a 2-hour time interval.

C#

```
// Define a time interval equal to two hours.
TimeSpan baseInterval = new TimeSpan( 2, 0, 0);

// Define an array of time intervals to compare with
// the base interval.
TimeSpan[] spans = {
    TimeSpan.FromSeconds(-2.5),
    TimeSpan.FromMinutes(20),
    TimeSpan.FromHours(1),
    TimeSpan.FromMinutes(90),
    baseInterval,
    TimeSpan.FromDays(.5),
    TimeSpan.FromDays(1)
};

// Compare the time intervals.
foreach (var span in spans) {
   int result = TimeSpan.Compare(baseInterval, span);
   Console.WriteLine("{0} {1} {2} (Compare returns {3})",
                     baseInterval,
                     result == 1 ? ">" : result == 0 ? "=" : "<",
                     span, result);
}

// The example displays the following output:
//       02:00:00 > -00:00:02.5000000 (Compare returns 1)
//       02:00:00 > 00:20:00 (Compare returns 1)
//       02:00:00 > 01:00:00 (Compare returns 1)
//       02:00:00 > 01:30:00 (Compare returns 1)
//       02:00:00 = 02:00:00 (Compare returns 0)
//       02:00:00 < 12:00:00 (Compare returns -1)
//       02:00:00 < 1.00:00:00 (Compare returns -1)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- CompareTo(Object)
- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or TimeSpan object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the specified object or TimeSpan object.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the specified object.
CompareTo(TimeSpan)	Compares this instance to a specified TimeSpan object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the TimeSpan object.

Remarks

The two overloads of the CompareTo method return a signed number that indicates the relative value of this instance and the `value` argument, as shown in the following table.

Expand table

Value	Description
Less than zero	This instance is shorter than `value`.
Zero	This instance is the same as `value`.
Greater than zero	This instance is longer than `value`.

CompareTo(Object)

Compares this instance to a specified object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the specified object.

```
C#

public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

One of the following values.

Expand table

Value	Description
-1	This instance is shorter than `value`.
0	This instance is equal to `value`.
1	This instance is longer than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a TimeSpan.

Examples

The following example compares several TimeSpan structures and other objects to a reference TimeSpan structure using the CompareTo method.

```
C#
```

```
// Example of the TimeSpan.CompareTo( Object ) and
// TimeSpan.Equals( Object ) methods.
using System;

class TSCompToEqualsObjDemo
{
    // Compare the TimeSpan to the Object parameters,
    // and display the Object parameters with the results.
    static void CompTimeSpanToObject( TimeSpan Left, object Right,
        string RightText )
    {
        Console.WriteLine( "{0,-33}{1}", "Object: " + RightText,
            Right );
        Console.WriteLine( "{0,-33}{1}", "Left.Equals( Object )",
            Left.Equals( Right ) );
        Console.Write( "{0,-33}", "Left.CompareTo( Object )" );

        // Catch the exception if CompareTo( ) throws one.
        try
        {
            Console.WriteLine( "{0}\n", Left.CompareTo( Right ) );
        }
        catch( Exception ex )
        {
            Console.WriteLine( "Error: {0}\n", ex.Message );
        }
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 0, 5, 0 );

        Console.WriteLine(
            "This example of the TimeSpan.Equals( Object ) " +
            "and \nTimeSpan.CompareTo( Object ) methods generates " +
            "the \nfollowing output by creating several different " +
            "TimeSpan \nobjects and comparing them with a " +
            "5-minute TimeSpan.\n" );
        Console.WriteLine( "{0,-33}{1}\n",
            "Left: TimeSpan( 0, 5, 0 )", Left );

        // Create objects to compare with a 5-minute TimeSpan.
        CompTimeSpanToObject( Left, new TimeSpan( 0, 0, 300 ),
            "TimeSpan( 0, 0, 300 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 0, 5, 1 ),
            "TimeSpan( 0, 5, 1 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 0, 5, -1 ),
            "TimeSpan( 0, 5, -1 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 3000000000 ),
            "TimeSpan( 3000000000 )" );
        CompTimeSpanToObject( Left, 3000000000L,
            "long 3000000000L" );
        CompTimeSpanToObject( Left, "00:05:00",
            "string \"00:05:00\"" );
```

```
    }
}

/*
This example of the TimeSpan.Equals( Object ) and
TimeSpan.CompareTo( Object ) methods generates the
following output by creating several different TimeSpan
objects and comparing them with a 5-minute TimeSpan.

Left: TimeSpan( 0, 5, 0 )        00:05:00

Object: TimeSpan( 0, 0, 300 )    00:05:00
Left.Equals( Object )            True
Left.CompareTo( Object )         0

Object: TimeSpan( 0, 5, 1 )      00:05:01
Left.Equals( Object )            False
Left.CompareTo( Object )         -1

Object: TimeSpan( 0, 5, -1 )     00:04:59
Left.Equals( Object )            False
Left.CompareTo( Object )         1

Object: TimeSpan( 3000000000 )   00:05:00
Left.Equals( Object )            True
Left.CompareTo( Object )         0

Object: long 3000000000L         3000000000
Left.Equals( Object )            False
Left.CompareTo( Object )         Error: Object must be of type TimeSpan.

Object: string "00:05:00"        00:05:00
Left.Equals( Object )            False
Left.CompareTo( Object )         Error: Object must be of type TimeSpan.
*/
```

Remarks

Any instance of TimeSpan, regardless of its value, is considered greater than `null`.

The `value` parameter must be an instance of TimeSpan or `null`; otherwise, an exception is thrown.

See also

- Object
- Compare(TimeSpan, TimeSpan)
- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CompareTo(TimeSpan)

Compares this instance to a specified TimeSpan object and returns an integer that indicates whether this instance is shorter than, equal to, or longer than the TimeSpan object.

C#

```
public int CompareTo (TimeSpan value);
```

Parameters

`value` TimeSpan

An object to compare to this instance.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Value	Description
A negative integer	This instance is shorter than `value`.
Zero	This instance is equal to `value`.
A positive integer	This instance is longer than `value`.

Implements

CompareTo(T)

Examples

The following example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;
```

```
// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
```

```
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223
```

```
The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the TimeSpan.CompareTo method because it does not have to convert the `value` parameter to an object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Divide Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Divide(TimeSpan)	Returns a new Double value that's the result of dividing this instance by `ts`.
Divide(Double)	Returns a new TimeSpan object whose value is the result of dividing this instance by the specified `divisor`.

Divide(TimeSpan)

Returns a new Double value that's the result of dividing this instance by `ts`.

C#

```
public double Divide (TimeSpan ts);
```

Parameters

`ts` TimeSpan

The value to be divided by.

Returns

Double

A new value that represents result of dividing this instance by the value of `ts`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Divide(Double)

Returns a new TimeSpan object whose value is the result of dividing this instance by the specified `divisor`.

```
C#
public TimeSpan Divide (double divisor);
```

Parameters

`divisor` Double

The value to be divided by.

Returns

TimeSpan

A new object that represents the value of this instance divided by the value of `divisor`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.Duration Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new TimeSpan object whose value is the absolute value of the current TimeSpan object.

C#

```
public TimeSpan Duration ();
```

Returns

TimeSpan

A new object whose value is the absolute value of the current TimeSpan object.

Exceptions

OverflowException

The value of this instance is TimeSpan.MinValue.

Examples

The following example applies the Duration method to several TimeSpan objects.

C#

```
// Example of the TimeSpan.Duration( ) and TimeSpan.Negate( ) methods,
// and the TimeSpan Unary Negation and Unary Plus operators.
using System;

class DuraNegaUnaryDemo
{
    const string dataFmt = "{0,22}{1,22}{2,22}" ;

    static void ShowDurationNegate( TimeSpan interval )
    {
        // Display the TimeSpan value and the results of the
```

```
        // Duration and Negate methods.
        Console.WriteLine( dataFmt,
            interval, interval.Duration( ), interval.Negate( ) );
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of TimeSpan.Duration( ), " +
            "TimeSpan.Negate( ), \nand the TimeSpan Unary " +
            "Negation and Unary Plus operators \n" +
            "generates the following output.\n" );
        Console.WriteLine( dataFmt,
            "TimeSpan", "Duration( )", "Negate( )" );
        Console.WriteLine( dataFmt,
            "--------", "-----------", "---------" );

        // Create TimeSpan objects and apply the Unary Negation
        // and Unary Plus operators to them.
        ShowDurationNegate( new TimeSpan( 1 ) );
        ShowDurationNegate( new TimeSpan( -1234567 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, 0, 10, -20, -30 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, -10, 20, -30, 40 ) );
        ShowDurationNegate(
            - new TimeSpan( 1, 10, 20, 40, 160 ) );
        ShowDurationNegate(
            - new TimeSpan( -10, -20, -30, -40, -50 ) );
    }
}

/*
This example of TimeSpan.Duration( ), TimeSpan.Negate( ),
and the TimeSpan Unary Negation and Unary Plus operators
generates the following output.

              TimeSpan           Duration( )             Negate( )
              --------           -----------             ---------
      00:00:00.0000001      00:00:00.0000001     -00:00:00.0000001
     -00:00:00.1234567      00:00:00.1234567      00:00:00.1234567
      00:09:39.9700000      00:09:39.9700000     -00:09:39.9700000
     -09:40:29.9600000      09:40:29.9600000      09:40:29.9600000
   -1.10:20:40.1600000    1.10:20:40.1600000    1.10:20:40.1600000
   10.20:30:40.0500000   10.20:30:40.0500000  -10.20:30:40.0500000
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Ticks

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two instances of TimeSpan are equal.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(TimeSpan)	Returns a value indicating whether this instance is equal to a specified TimeSpan object.
Equals(TimeSpan, TimeSpan)	Returns a value that indicates whether two specified instances of TimeSpan are equal.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? value);
```

Parameters

`value` Object

An object to compare with this instance.

Returns

Boolean

`true` if `value` is a TimeSpan object that represents the same time interval as the current TimeSpan structure; otherwise, `false`.

Examples

The following example compares several TimeSpan and other objects to a reference TimeSpan using the Equals method.

C#

```
// Example of the TimeSpan.CompareTo( Object ) and
// TimeSpan.Equals( Object ) methods.
using System;

class TSCompToEqualsObjDemo
{
    // Compare the TimeSpan to the Object parameters,
    // and display the Object parameters with the results.
    static void CompTimeSpanToObject( TimeSpan Left, object Right,
        string RightText )
    {
        Console.WriteLine( "{0,-33}{1}", "Object: " + RightText,
            Right );
        Console.WriteLine( "{0,-33}{1}", "Left.Equals( Object )",
            Left.Equals( Right ) );
        Console.Write( "{0,-33}", "Left.CompareTo( Object )" );

        // Catch the exception if CompareTo( ) throws one.
        try
        {
            Console.WriteLine( "{0}\n", Left.CompareTo( Right ) );
        }
        catch( Exception ex )
        {
            Console.WriteLine( "Error: {0}\n", ex.Message );
        }
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 0, 5, 0 );

        Console.WriteLine(
            "This example of the TimeSpan.Equals( Object ) " +
            "and \nTimeSpan.CompareTo( Object ) methods generates " +
            "the \nfollowing output by creating several different " +
            "TimeSpan \nobjects and comparing them with a " +
            "5-minute TimeSpan.\n" );
        Console.WriteLine( "{0,-33}{1}\n",
            "Left: TimeSpan( 0, 5, 0 )", Left );

        // Create objects to compare with a 5-minute TimeSpan.
```

```
        CompTimeSpanToObject( Left, new TimeSpan( 0, 0, 300 ),
            "TimeSpan( 0, 0, 300 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 0, 5, 1 ),
            "TimeSpan( 0, 5, 1 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 0, 5, -1 ),
            "TimeSpan( 0, 5, -1 )" );
        CompTimeSpanToObject( Left, new TimeSpan( 3000000000 ),
            "TimeSpan( 3000000000 )" );
        CompTimeSpanToObject( Left, 3000000000L,
            "long 3000000000L" );
        CompTimeSpanToObject( Left, "00:05:00",
            "string \"00:05:00\"" );
    }
}

/*
This example of the TimeSpan.Equals( Object ) and
TimeSpan.CompareTo( Object ) methods generates the
following output by creating several different TimeSpan
objects and comparing them with a 5-minute TimeSpan.

Left: TimeSpan( 0, 5, 0 )        00:05:00

Object: TimeSpan( 0, 0, 300 )    00:05:00
Left.Equals( Object )            True
Left.CompareTo( Object )         0

Object: TimeSpan( 0, 5, 1 )      00:05:01
Left.Equals( Object )            False
Left.CompareTo( Object )         -1

Object: TimeSpan( 0, 5, -1 )     00:04:59
Left.Equals( Object )            False
Left.CompareTo( Object )         1

Object: TimeSpan( 3000000000 )   00:05:00
Left.Equals( Object )            True
Left.CompareTo( Object )         0

Object: long 3000000000L         3000000000
Left.Equals( Object )            False
Left.CompareTo( Object )         Error: Object must be of type TimeSpan.

Object: string "00:05:00"        00:05:00
Left.Equals( Object )            False
Left.CompareTo( Object )         Error: Object must be of type TimeSpan.
*/
```

See also

- Object
- Compare(TimeSpan, TimeSpan)

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(TimeSpan)

Returns a value indicating whether this instance is equal to a specified TimeSpan object.

```
C#

public bool Equals (TimeSpan obj);
```

Parameters

`obj` TimeSpan

An object to compare with this instance.

Returns

Boolean

`true` if `obj` represents the same time interval as this instance; otherwise, `false`.

Implements

Equals(T)

Examples

The following example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
```

```
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
```

```
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(TimeSpan, TimeSpan)

Returns a value that indicates whether two specified instances of TimeSpan are equal.

C#

```
public static bool Equals (TimeSpan t1, TimeSpan t2);
```

Parameters

`t1` TimeSpan

The first time interval to compare.

`t2` TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the values of `t1` and `t2` are equal; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan object using the static Equals method.

C#

```
// Example of the TimeSpan.Compare( TimeSpan, TimeSpan ) and
// TimeSpan.Equals( TimeSpan, TimeSpan ) methods.
using System;

class TSCompareEqualsDemo
{
    const string dataFmt = "{0,-38}{1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "TimeSpan.Equals( Left, Right )",
            TimeSpan.Equals( Left, Right ) );
        Console.WriteLine( dataFmt,
            "TimeSpan.Compare( Left, Right )",
            TimeSpan.Compare( Left, Right ) );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan.Equals( TimeSpan, Time" +
            "Span ) and \nTimeSpan.Compare( TimeSpan, TimeSpan ) " +
            "methods generates the \nfollowing output by creating " +
            "several different TimeSpan \nobjects and comparing " +
            "them with a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt, "Left: TimeSpan( 2, 0, 0 )",
            Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
```

```
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )");
        CompareTimeSpans( Left, new TimeSpan( 72000000000 ),
            "TimeSpan( 72000000000 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan.Equals( TimeSpan, TimeSpan ) and
TimeSpan.Compare( TimeSpan, TimeSpan ) methods generates the
following output by creating several different TimeSpan
objects and comparing them with a 2-hour TimeSpan.

Left: TimeSpan( 2, 0, 0 )             02:00:00

Right: TimeSpan( 0, 120, 0 )          02:00:00
TimeSpan.Equals( Left, Right )        True
TimeSpan.Compare( Left, Right )       0

Right: TimeSpan( 2, 0, 1 )            02:00:01
TimeSpan.Equals( Left, Right )        False
TimeSpan.Compare( Left, Right )       -1

Right: TimeSpan( 2, 0, -1 )           01:59:59
TimeSpan.Equals( Left, Right )        False
TimeSpan.Compare( Left, Right )       1

Right: TimeSpan( 72000000000 )        02:00:00
TimeSpan.Equals( Left, Right )        True
TimeSpan.Compare( Left, Right )       0

Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
TimeSpan.Equals( Left, Right )        True
TimeSpan.Compare( Left, Right )       0
*/
```

See also

- Boolean
- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromDays(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of days, where the specification is accurate to the nearest millisecond.

```
C#
public static TimeSpan FromDays (double value);
```

Parameters

`value` Double

A number of days, accurate to the nearest millisecond.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

`value` is PositiveInfinity.

-or-

`value` is NegativeInfinity.

ArgumentException

`value` is equal to NaN.

Examples

The following example creates several TimeSpan objects using the FromDays method.

C#

```
// Example of the TimeSpan.FromDays( double ) method.
using System;

class FromDaysDemo
{
    static void GenTimeSpanFromDays( double days )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of days.
        TimeSpan interval = TimeSpan.FromDays( days );
        string   timeInterval = interval.ToString( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", days, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromDays( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromDays", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "--------", "--------" );

        GenTimeSpanFromDays( 0.000000006 );
        GenTimeSpanFromDays( 0.000000017 );
        GenTimeSpanFromDays( 0.000123456 );
        GenTimeSpanFromDays( 1.234567898 );
        GenTimeSpanFromDays( 12345.678987654 );
        GenTimeSpanFromDays( 0.000011574 );
        GenTimeSpanFromDays( 0.000694444 );
        GenTimeSpanFromDays( 0.041666666 );
        GenTimeSpanFromDays( 1 );
        GenTimeSpanFromDays( 20.84745602 );
    }
}

/*
This example of TimeSpan.FromDays( double )
generates the following output.
```

```
              FromDays          TimeSpan
              --------          --------
                 6E-09          00:00:00.0010000
               1.7E-08          00:00:00.0010000
           0.000123456          00:00:10.6670000
           1.234567898        1.05:37:46.6660000
       12345.678987654    12345.16:17:44.5330000
            1.1574E-05          00:00:01
           0.000694444          00:01:00
           0.041666666          01:00:00
                     1        1.00:00:00
           20.84745602       20.20:20:20.2000000
*/
```

Remarks

The `value` parameter is converted to milliseconds, which is converted to ticks, and that number of ticks is used to initialize the new TimeSpan. Therefore, `value` will only be considered accurate to the nearest millisecond. Note that, because of the loss of precision of the Double data type, this conversion can cause an OverflowException for values that are near to but still in the range of either MinValue or MaxValue. For example, this causes an OverflowException in the following attempt to instantiate a TimeSpan object.

C#

```
// The following throws an OverflowException at runtime
TimeSpan maxSpan = TimeSpan.FromDays(TimeSpan.MaxValue.TotalDays);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- FromTicks(Int64)
- FromMilliseconds(Double)
- FromSeconds(Double)
- FromMinutes(Double)
- FromHours(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromHours(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of hours, where the specification is accurate to the nearest millisecond.

```
C#
public static TimeSpan FromHours (double value);
```

Parameters

`value` Double

A number of hours accurate to the nearest millisecond.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

`value` is PositiveInfinity.

-or-

`value` is NegativeInfinity.

ArgumentException

`value` is equal to NaN.

Examples

The following example creates several TimeSpan objects using the FromHours method.

C#

```
// Example of the TimeSpan.FromHours( double ) method.
using System;

class FromHoursDemo
{
    static void GenTimeSpanFromHours( double hours )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of hours.
        TimeSpan    interval = TimeSpan.FromHours( hours );
        string      timeInterval = interval.ToString( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", hours, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromHours( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromHours", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "---------", "--------" );

        GenTimeSpanFromHours( 0.0000002 );
        GenTimeSpanFromHours( 0.0000003 );
        GenTimeSpanFromHours( 0.0012345 );
        GenTimeSpanFromHours( 12.3456789 );
        GenTimeSpanFromHours( 123456.7898765 );
        GenTimeSpanFromHours( 0.0002777 );
        GenTimeSpanFromHours( 0.0166666 );
        GenTimeSpanFromHours( 1 );
        GenTimeSpanFromHours( 24 );
        GenTimeSpanFromHours( 500.3389445 );
    }
}

/*
This example of TimeSpan.FromHours( double )
generates the following output.
```

```
            FromHours          TimeSpan
            ---------          --------
                2E-07          00:00:00.0010000
                3E-07          00:00:00.0010000
            0.0012345          00:00:04.4440000
           12.3456789          12:20:44.4440000
       123456.7898765     5144.00:47:23.5550000
            0.0002777          00:00:01
            0.0166666          00:01:00
                    1          01:00:00
                   24        1.00:00:00
          500.3389445       20.20:20:20.2000000
*/
```

Remarks

The `value` parameter is converted to milliseconds, which is converted to ticks, and that number of ticks is used to initialize the new TimeSpan. Therefore, `value` will only be considered accurate to the nearest millisecond. Note that, because of the loss of precision of the Double data type, this conversion can generate an OverflowException for values that are near to but still in the range of either MinValue or MaxValue. For example, this causes an OverflowException in the following attempt to instantiate a TimeSpan object.

C#

```
// The following throws an OverflowException at runtime
TimeSpan maxSpan = TimeSpan.FromHours(TimeSpan.MaxValue.TotalHours);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- FromTicks(Int64)
- FromMilliseconds(Double)
- FromSeconds(Double)
- FromMinutes(Double)
- FromDays(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromMicroseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of microseconds.

```
C#
public static TimeSpan FromMicroseconds (double value);
```

Parameters

`value` Double

A number of microseconds.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than MinValue or greater than MaxValue.

-or-

`value` is PositiveInfinity

-or-

`value` is NegativeInfinity

ArgumentException

`value` is equal to NaN.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.FromMilliseconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of milliseconds.

```
C#
public static TimeSpan FromMilliseconds (double value);
```

Parameters

`value` Double

A number of milliseconds.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

`value` is PositiveInfinity.

-or-

`value` is NegativeInfinity.

ArgumentException

`value` is equal to NaN.

Examples

The following example creates several TimeSpan objects by using the FromMilliseconds method.

C#

```
// Example of the TimeSpan.FromMilliseconds( double ) method.
using System;

class FromMillisecDemo
{
    static void GenTimeSpanFromMillisec( Double millisec )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of milliseconds.
        TimeSpan    interval = TimeSpan.FromMilliseconds( millisec );
        string      timeInterval = interval.ToString( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", millisec, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromMilliseconds( " +
            "double )\ngenerates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromMilliseconds", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "----------------", "--------" );

        GenTimeSpanFromMillisec( 1 );
        GenTimeSpanFromMillisec( 1.5 );
        GenTimeSpanFromMillisec( 12345.6 );
        GenTimeSpanFromMillisec( 123456789.8 );
        GenTimeSpanFromMillisec( 1234567898765.4 );
        GenTimeSpanFromMillisec( 1000 );
        GenTimeSpanFromMillisec( 60000 );
        GenTimeSpanFromMillisec( 3600000 );
        GenTimeSpanFromMillisec( 86400000 );
        GenTimeSpanFromMillisec( 1801220200 );
    }
}

/*
This example of TimeSpan.FromMilliseconds( double )
```

```
generates the following output.

     FromMilliseconds          TimeSpan
     ----------------          --------
                    1          00:00:00.0010000
                  1.5          00:00:00.0020000
              12345.6          00:00:12.3460000
          123456789.8        1.10:17:36.7900000
       1234567898765.4    14288.23:31:38.7650000
                 1000          00:00:01
                60000          00:01:00
              3600000          01:00:00
             86400000        1.00:00:00
           1801220200       20.20:20:20.2000000
*/
```

Remarks

The `value` parameter is converted to ticks, and that number of ticks is used to initialize the new TimeSpan. Therefore, `value` will only be considered accurate to the nearest millisecond. Note that, because of the loss of precision of the Double data type, this conversion can generate an OverflowException for values that are near to but still in the range of either MinValue or MaxValue. For example, this causes an OverflowException in the following attempt to instantiate a TimeSpan object.

C#

```
// The following throws an OverflowException at runtime
TimeSpan maxSpan =
TimeSpan.FromMilliseconds(TimeSpan.MaxValue.TotalMilliseconds);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- FromTicks(Int64)
- FromSeconds(Double)
- FromMinutes(Double)
- FromHours(Double)
- FromDays(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromMinutes(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of minutes, where the specification is accurate to the nearest millisecond.

```
C#

public static TimeSpan FromMinutes (double value);
```

Parameters

`value` Double

A number of minutes, accurate to the nearest millisecond.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

`value` is PositiveInfinity.

-or-

`value` is NegativeInfinity.

ArgumentException

`value` is equal to NaN.

Examples

The following example creates several TimeSpan objects using the FromMinutes method.

C#

```
// Example of the TimeSpan.FromMinutes( double ) method.
using System;

class FromMinutesDemo
{
    static void GenTimeSpanFromMinutes( double minutes )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of minutes.
        TimeSpan    interval = TimeSpan.FromMinutes( minutes );
        string      timeInterval = interval.ToString( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", minutes, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromMinutes( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromMinutes", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "-----------", "--------" );

        GenTimeSpanFromMinutes( 0.00001 );
        GenTimeSpanFromMinutes( 0.00002 );
        GenTimeSpanFromMinutes( 0.12345 );
        GenTimeSpanFromMinutes( 1234.56789 );
        GenTimeSpanFromMinutes( 12345678.98765 );
        GenTimeSpanFromMinutes( 0.01666 );
        GenTimeSpanFromMinutes( 1 );
        GenTimeSpanFromMinutes( 60 );
        GenTimeSpanFromMinutes( 1440 );
        GenTimeSpanFromMinutes( 30020.33667 );
    }
}
```

```
/*
This example of TimeSpan.FromMinutes( double )
generates the following output.

          FromMinutes          TimeSpan
          -----------          --------
                1E-05          00:00:00.0010000
                2E-05          00:00:00.0010000
              0.12345          00:00:07.4070000
           1234.56789          20:34:34.0730000
       12345678.98765     8573.09:18:59.2590000
              0.01666          00:00:01
                    1          00:01:00
                   60          01:00:00
                 1440        1.00:00:00
          30020.33667       20.20:20:20.2000000
*/
```

Remarks

The `value` parameter is converted to milliseconds, which is converted to ticks, and that number of ticks is used to initialize the new TimeSpan. Therefore, `value` will only be considered accurate to the nearest millisecond. Note that, because of the loss of precision of the Double data type, this conversion can generate an OverflowException for values that are near to but still in the range of either MinValue or MaxValue. For example, this causes an OverflowException in the following attempt to instantiate a TimeSpan object.

C#

```
// The following throws an OverflowException at runtime
TimeSpan maxSpan = TimeSpan.FromMinutes(TimeSpan.MaxValue.TotalMinutes);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- FromTicks(Int64)
- FromMilliseconds(Double)
- FromSeconds(Double)
- FromHours(Double)
- FromDays(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromSeconds(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified number of seconds, where the specification is accurate to the nearest millisecond.

```
C#

public static TimeSpan FromSeconds (double value);
```

Parameters

`value` Double

A number of seconds, accurate to the nearest millisecond.

Returns

TimeSpan

An object that represents `value`.

Exceptions

OverflowException

`value` is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

`value` is PositiveInfinity.

-or-

`value` is NegativeInfinity.

ArgumentException

`value` is equal to NaN.

Examples

The following example creates several TimeSpan objects using the FromSeconds method.

C#

```
// Example of the TimeSpan.FromSeconds( double ) method.
using System;

class FromSecondsDemo
{
    static void GenTimeSpanFromSeconds( double seconds )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of seconds.
        TimeSpan    interval = TimeSpan.FromSeconds( seconds );
        string      timeInterval = interval.ToString( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", seconds, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromSeconds( double )\n" +
            "generates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromSeconds", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "-----------", "--------" );

        GenTimeSpanFromSeconds( 0.001 );
        GenTimeSpanFromSeconds( 0.0015 );
        GenTimeSpanFromSeconds( 12.3456 );
        GenTimeSpanFromSeconds( 123456.7898 );
        GenTimeSpanFromSeconds( 1234567898.7654 );
        GenTimeSpanFromSeconds( 1 );
        GenTimeSpanFromSeconds( 60 );
        GenTimeSpanFromSeconds( 3600 );
        GenTimeSpanFromSeconds( 86400 );
        GenTimeSpanFromSeconds( 1801220.2 );
    }
}
```

```
/*
This example of TimeSpan.FromSeconds( double )
generates the following output.

          FromSeconds          TimeSpan
          -----------          --------
                0.001          00:00:00.0010000
               0.0015          00:00:00.0020000
              12.3456          00:00:12.3460000
          123456.7898        1.10:17:36.7900000
      1234567898.7654    14288.23:31:38.7650000
                    1          00:00:01
                   60          00:01:00
                 3600          01:00:00
                86400        1.00:00:00
            1801220.2       20.20:20:20.2000000
*/
```

Remarks

The `value` parameter is converted to milliseconds, which is converted to ticks, and that number of ticks is used to initialize the new TimeSpan. Therefore, `value` will only be considered accurate to the nearest millisecond. Note that, because of the loss of precision of the Double data type, this can generate an OverflowException for values that are near but still in the range of either MinValue or MaxValue, This is the cause of an OverflowException, for example, in the following attempt to instantiate a TimeSpan object.

C#

```
// The following throws an OverflowException at runtime
TimeSpan maxSpan = TimeSpan.FromSeconds(TimeSpan.MaxValue.TotalSeconds);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Double
- FromTicks(Int64)
- FromMilliseconds(Double)
- FromMinutes(Double)
- FromHours(Double)
- FromDays(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.FromTicks(Int64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan that represents a specified time, where the specification is in units of ticks.

C#

```
public static TimeSpan FromTicks (long value);
```

Parameters

`value` Int64

A number of ticks that represent a time.

Returns

TimeSpan

An object that represents `value`.

Examples

The following example creates several TimeSpan objects using the FromTicks method.

C#

```
// Example of the TimeSpan.FromTicks( long ) method.
using System;

class FromTicksDemo
{
    static void GenTimeSpanFromTicks( long ticks )
    {
        // Create a TimeSpan object and TimeSpan string from
        // a number of ticks.
        TimeSpan    interval = TimeSpan.FromTicks( ticks );
        string      timeInterval = interval.ToString( );
```

```
        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";

        Console.WriteLine( "{0,21}{1,26}", ticks, timeInterval );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.FromTicks( long )\n" +
            "generates the following output.\n" );
        Console.WriteLine( "{0,21}{1,18}",
            "FromTicks", "TimeSpan" );
        Console.WriteLine( "{0,21}{1,18}",
            "---------", "--------" );

        GenTimeSpanFromTicks( 1 );
        GenTimeSpanFromTicks( 12345 );
        GenTimeSpanFromTicks( 123456789 );
        GenTimeSpanFromTicks( 1234567898765 );
        GenTimeSpanFromTicks( 12345678987654321 );
        GenTimeSpanFromTicks( 10000000 );
        GenTimeSpanFromTicks( 600000000 );
        GenTimeSpanFromTicks( 36000000000 );
        GenTimeSpanFromTicks( 864000000000 );
        GenTimeSpanFromTicks( 18012202000000 );
    }
}

/*
This example of TimeSpan.FromTicks( long )
generates the following output.

            FromTicks          TimeSpan
            ---------          --------
                    1          00:00:00.0000001
                12345          00:00:00.0012345
            123456789          00:00:12.3456789
        1234567898765        1.10:17:36.7898765
    12345678987654321    14288.23:31:38.7654321
             10000000          00:00:01
            600000000          00:01:00
          36000000000          01:00:00
         864000000000        1.00:00:00
       18012202000000       20.20:20:20.2000000
*/
```

Remarks

This is a convenience method with the same behavior as the TimeSpan.TimeSpan(Int64) constructor. A single tick represents one hundred nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Int64
- FromMilliseconds(Double)
- FromSeconds(Double)
- FromMinutes(Double)
- FromHours(Double)
- FromDays(Double)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Examples

The following example generates the hash codes of several TimeSpan objects using the GetHashCode method.

C#

```
// Example for the TimeSpan.GetHashCode( ) method.
using System;

class GetHashCode
{
    static void DisplayHashCode( TimeSpan interval )
    {
        // Create a hash code and a string representation of
        // the TimeSpan parameter.
        string  timeInterval = interval.ToString( );
        int     hashCode = interval.GetHashCode( );

        // Pad the end of the TimeSpan string with spaces if it
        // does not contain milliseconds.
        int pIndex = timeInterval.IndexOf( ':' );
        pIndex = timeInterval.IndexOf( '.', pIndex );
        if( pIndex < 0 )   timeInterval += "        ";
```

```
        Console.WriteLine( "{0,22}   0x{1:X8}, {1}",
            timeInterval, hashCode );
    }

    static void Main( )
    {
        Console.WriteLine(
            "This example of TimeSpan.GetHashCode( ) generates " +
            "the following \noutput, which displays " +
            "the hash codes of representative TimeSpan \n" +
            "objects in hexadecimal and decimal formats.\n" );
        Console.WriteLine( "{0,22}   {1,10}",
            "TimeSpan        ", "Hash Code" );
        Console.WriteLine( "{0,22}   {1,10}",
            "--------        ", "---------" );

        DisplayHashCode( new TimeSpan( 0 ) );
        DisplayHashCode( new TimeSpan( 1 ) );
        DisplayHashCode( new TimeSpan( 0, 0, 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 0, 1, 0 ) );
        DisplayHashCode( new TimeSpan( 1, 0, 0 ) );
        DisplayHashCode( new TimeSpan( 36000000001 ) );
        DisplayHashCode( new TimeSpan( 0, 1, 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 1, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 1, 0, 0, 0 ) );
        DisplayHashCode( new TimeSpan( 864000000001 ) );
        DisplayHashCode( new TimeSpan( 1, 0, 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 1, 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 100, 0, 0, 0 ) );
        DisplayHashCode( new TimeSpan( 100, 0, 0, 0, 1 ) );
        DisplayHashCode( new TimeSpan( 100, 0, 0, 1 ) );
    }
}

/*
This example of TimeSpan.GetHashCode( ) generates the following
output, which displays the hash codes of representative TimeSpan
objects in hexadecimal and decimal formats.

      TimeSpan            Hash Code
      --------            ---------
      00:00:00           0x00000000, 0
      00:00:00.0000001   0x00000001, 1
      00:00:00.0010000   0x00002710, 10000
      00:00:01           0x00989680, 10000000
      00:01:00           0x23C34600, 600000000
      01:00:00           0x61C46808, 1640261640
      01:00:00.0000001   0x61C46809, 1640261641
      01:00:00.0010000   0x61C48F18, 1640271640
      01:00:01           0x625CFE88, 1650261640
    1.00:00:00           0x2A69C0C9, 711573705
    1.00:00:00.0000001   0x2A69C0C8, 711573704
    1.00:00:00.0010000   0x2A69E7D9, 711583705
```

```
    1.00:00:01           0x2B025649, 721573449
  100.00:00:00           0x914F4E94, -1857073516
  100.00:00:00.0010000   0x914F6984, -1857066620
  100.00:00:01           0x91E7D814, -1847076844
*/
```

Remarks

Two TimeSpan objects might have the same hash code even though they represent different time values.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Multiply(Double) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new TimeSpan object which value is the result of multiplication of this instance and the specified `factor`.

```
C#
public TimeSpan Multiply (double factor);
```

Parameters

`factor` Double

The value to be multiplied by.

Returns

TimeSpan

A new object that represents the value of this instance multiplied by the value of `factor`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeSpan.Negate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new TimeSpan object whose value is the negated value of this instance.

C#

```
public TimeSpan Negate ();
```

Returns

TimeSpan

A new object with the same numeric value as this instance, but with the opposite sign.

Exceptions

OverflowException

The negated value of this instance cannot be represented by a TimeSpan; that is, the value of this instance is TimeSpan.MinValue.

Examples

The following example applies the Negate method to several TimeSpan objects.

C#

```
// Example of the TimeSpan.Duration( ) and TimeSpan.Negate( ) methods,
// and the TimeSpan Unary Negation and Unary Plus operators.
using System;

class DuraNegaUnaryDemo
{
    const string dataFmt = "{0,22}{1,22}{2,22}" ;

    static void ShowDurationNegate( TimeSpan interval )
    {
        // Display the TimeSpan value and the results of the
```

```
        // Duration and Negate methods.
        Console.WriteLine( dataFmt,
            interval, interval.Duration( ), interval.Negate( ) );
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of TimeSpan.Duration( ), " +
            "TimeSpan.Negate( ), \nand the TimeSpan Unary " +
            "Negation and Unary Plus operators \n" +
            "generates the following output.\n" );
        Console.WriteLine( dataFmt,
            "TimeSpan", "Duration( )", "Negate( )" );
        Console.WriteLine( dataFmt,
            "--------", "-----------", "---------" );

        // Create TimeSpan objects and apply the Unary Negation
        // and Unary Plus operators to them.
        ShowDurationNegate( new TimeSpan( 1 ) );
        ShowDurationNegate( new TimeSpan( -1234567 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, 0, 10, -20, -30 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, -10, 20, -30, 40 ) );
        ShowDurationNegate(
            - new TimeSpan( 1, 10, 20, 40, 160 ) );
        ShowDurationNegate(
            - new TimeSpan( -10, -20, -30, -40, -50 ) );
    }
}

/*
This example of TimeSpan.Duration( ), TimeSpan.Negate( ),
and the TimeSpan Unary Negation and Unary Plus operators
generates the following output.

              TimeSpan           Duration( )             Negate( )
              --------           -----------             ---------
      00:00:00.0000001      00:00:00.0000001     -00:00:00.0000001
     -00:00:00.1234567      00:00:00.1234567      00:00:00.1234567
      00:09:39.9700000      00:09:39.9700000     -00:09:39.9700000
     -09:40:29.9600000      09:40:29.9600000      09:40:29.9600000
   -1.10:20:40.1600000    1.10:20:40.1600000    1.10:20:40.1600000
   10.20:30:40.0500000   10.20:30:40.0500000  -10.20:30:40.0500000
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a time interval to its TimeSpan equivalent.

Overloads

Expand table

Parse(String)	Converts the string representation of a time interval to its TimeSpan equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific format information.
Parse(String, IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-specific format information.

Parse(String)

Converts the string representation of a time interval to its TimeSpan equivalent.

C#

```
public static TimeSpan Parse (string s);
```

Parameters

s String

A string that specifies the time interval to convert.

Returns

TimeSpan

A time interval that corresponds to `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` has an invalid format.

OverflowException

`s` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components is outside its valid range.

Examples

The following example uses the Parse method to convert each element in a string array to a TimeSpan value. It changes the current system culture to Croatian - Croatia ("hr-HR") and English - United States ("en-US") to illustrate how the current system culture affects the parsing operation.

C#

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
   public static void Main()
   {
      string[] values = { "6", "6:12", "6:12:14", "6:12:14:45",
                          "6.12:14:45", "6:12:14:45.3448",
                          "6:12:14:45,3448", "6:34:14:45" };
      string[] cultureNames = { "hr-HR", "en-US"};

      // Change the current culture.
      foreach (string cultureName in cultureNames)
      {
```

```
            Thread.CurrentThread.CurrentCulture = new
CultureInfo(cultureName);
            Console.WriteLine("Current Culture: {0}",
                              Thread.CurrentThread.CurrentCulture.Name);
            foreach (string value in values)
            {
               try {
                  TimeSpan ts = TimeSpan.Parse(value);
                  Console.WriteLine("{0} --> {1}", value, ts.ToString("c"));
               }
               catch (FormatException) {
                  Console.WriteLine("{0}: Bad Format", value);
               }
               catch (OverflowException) {
                  Console.WriteLine("{0}: Overflow", value);
               }
            }
            Console.WriteLine();
         }
      }
   }
// The example displays the following output:
//    Current Culture: hr-HR
//    6 --> 6.00:00:00
//    6:12 --> 06:12:00
//    6:12:14 --> 06:12:14
//    6:12:14:45 --> 6.12:14:45
//    6.12:14:45 --> 6.12:14:45
//    6:12:14:45.3448: Bad Format
//    6:12:14:45,3448 --> 6.12:14:45.3448000
//    6:34:14:45: Overflow
//
//    Current Culture: en-US
//    6 --> 6.00:00:00
//    6:12 --> 06:12:00
//    6:12:14 --> 06:12:14
//    6:12:14:45 --> 6.12:14:45
//    6.12:14:45 --> 6.12:14:45
//    6:12:14:45.3448 --> 6.12:14:45.3448000
//    6:12:14:45,3448: Bad Format
//    6:34:14:45: Overflow
```

Remarks

For more information about this API, see Supplemental API remarks for TimeSpan.Parse.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific format information.

C#

```
public static TimeSpan Parse (ReadOnlySpan<char> input, IFormatProvider?
formatProvider = default);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent the time interval to convert.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `formatProvider`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String, IFormatProvider)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-specific format information.

C#

```
public static TimeSpan Parse (string input, IFormatProvider?
formatProvider);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `formatProvider`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`input` is `null`.

FormatException

`input` has an invalid format.

OverflowException

`input` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components in `input` is outside its valid range.

Examples

The following example defines an array of CultureInfo objects, and uses each object in calls to the Parse(String, IFormatProvider) method to parse the elements in a string array. The example illustrates how the conventions of a specific culture influence the formatting operation.

C#

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      string[] values = { "6", "6:12", "6:12:14", "6:12:14:45",
                          "6.12:14:45", "6:12:14:45.3448",
                          "6:12:14:45,3448", "6:34:14:45" };
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("ru-RU"),
                                 CultureInfo.InvariantCulture };

      string header = String.Format("{0,-17}", "String");
      foreach (CultureInfo culture in cultures)
         header += culture.Equals(CultureInfo.InvariantCulture) ?
                      String.Format("{0,20}", "Invariant") :
                      String.Format("{0,20}", culture.Name);
      Console.WriteLine(header);
      Console.WriteLine();

      foreach (string value in values)
      {
         Console.Write("{0,-17}", value);
         foreach (CultureInfo culture in cultures)
         {
            try {
               TimeSpan ts = TimeSpan.Parse(value, culture);
               Console.Write("{0,20}", ts.ToString("c"));
            }
```

```
            catch (FormatException) {
               Console.Write("{0,20}", "Bad Format");
            }
            catch (OverflowException) {
               Console.Write("{0,20}", "Overflow");
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    String                          en-US               ru-RU
Invariant
//
//    6                         6.00:00:00          6.00:00:00
6.00:00:00
//    6:12                        06:12:00            06:12:00
06:12:00
//    6:12:14                     06:12:14            06:12:14
06:12:14
//    6:12:14:45                6.12:14:45          6.12:14:45
6.12:14:45
//    6.12:14:45                6.12:14:45          6.12:14:45
6.12:14:45
//    6:12:14:45.3448    6.12:14:45.3448000          Bad Format
6.12:14:45.3448000
//    6:12:14:45,3448           Bad Format  6.12:14:45.3448000
Bad Format
//    6:34:14:45                  Overflow            Overflow
Overflow
```

Remarks

This method tries to parse `input` by using each of the culture-specific formats for the culture specified by `formatProvider`.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that

provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

For more information about this API, see Supplemental API remarks for System.TimeSpan.Parse.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.ParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a time interval to its TimeSpan equivalent. The format of the string representation must match a specified format exactly.

Overloads

Expand table

ParseExact(String, String, IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles. The format of the string representation must match the specified format exactly.
ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String[], IFormatProvider, TimeSpanStyles)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String[], IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified array of format strings and culture-specific format information. The format of the string representation must match one of the specified formats exactly.
ParseExact(String, String, IFormatProvider)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles)	Converts the char span of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

ParseExact(String, String, IFormatProvider, TimeSpanStyles)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles. The format of the string representation must match the specified format exactly.

C#

```
public static TimeSpan ParseExact (string input, string format,
IFormatProvider? formatProvider, System.Globalization.TimeSpanStyles
styles);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`format` String

A standard or custom format string that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`styles` TimeSpanStyles

A bitwise combination of enumeration values that defines the style elements that may be present in `input`.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `format`, `formatProvider`, and `styles`.

Exceptions

ArgumentException

`styles` is an invalid TimeSpanStyles value.

ArgumentNullException

`input` is `null`.

FormatException

`input` has an invalid format.

OverflowException

`input` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components in `input` is outside its valid range.

Examples

The following example uses the ParseExact(String, String, IFormatProvider) method to parse several string representations of time intervals using various format strings and cultures. It also uses the TimeSpanStyles.AssumeNegative value to interpret each string as a negative time interval. The output from the example illustrates that the TimeSpanStyles.AssumeNegative style affects the return value only when it is used with custom format strings.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string intervalString, format;
      TimeSpan interval;
      CultureInfo culture = null;

      // Parse hour:minute value with custom format specifier.
      intervalString = "17:14";
      format = "h\\:mm";
      culture = CultureInfo.CurrentCulture;
```

```
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse hour:minute:second value with "g" specifier.
      intervalString = "17:14:48";
      format = "g";
      culture = CultureInfo.InvariantCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse hours:minute.second value with custom format specifier.
      intervalString = "17:14:48.153";
      format = @"h\:mm\:ss\.fff";
      culture = null;
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with "G" specifier
      // and current (en-US) culture.
```

```
      intervalString = "3:17:14:48.153";
      format = "G";
      culture = CultureInfo.CurrentCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with a custom format
specifier.
      intervalString = "3:17:14:48.153";
      format = @"d\:hh\:mm\:ss\.fff";
      culture = null;
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48,153";
      format = "G";
      culture = new CultureInfo("fr-FR");
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        culture,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
```

```
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse a single number using the "c" standard format string.
      intervalString = "12";
      format = "c";
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        null,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse a single number using the "%h" custom format string.
      format = "%h";
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        null,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse a single number using the "%s" custom format string.
      format = "%s";
      try {
         interval = TimeSpan.ParseExact(intervalString, format,
                                        null,
TimeSpanStyles.AssumeNegative);
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }
   }
```

```
}
// The example displays the following output:
//    '17:14' (h\:mm) --> -17:14:00
//    '17:14:48' (g) --> 17:14:48
//    '17:14:48.153' (h\:mm\:ss\.fff) --> -17:14:48.1530000
//    '3:17:14:48.153' (G) --> 3.17:14:48.1530000
//    '3:17:14:48.153' (d\:hh\:mm\:ss\.fff) --> -3.17:14:48.1530000
//    '3:17:14:48,153' (G) --> 3.17:14:48.1530000
//    '12' (c) --> 12.00:00:00
//    '12' (%h) --> -12:00:00
//    '12' (%s) --> -00:00:12
```

Remarks

The ParseExact method parses the string representation of a time interval, which must be in the format defined by the `format` parameter, except that leading and trailing white-space characters are ignored. Because `input` must conform to the format of `format` exactly, you should always use exception handling when converting a string input by the user to a time interval. If you prefer not to use exception handling, you can call the TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan) method instead.

The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.

> **Important**
>
> The **ParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if `format` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if `format` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object

returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.

- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

The `styles` parameter affects the interpretation of strings that are parsed using custom format strings. It determines whether `input` is interpreted as a negative time interval only if a negative sign is present (TimeSpanStyles.None), or whether it is always interpreted as a negative time interval (TimeSpanStyles.AssumeNegative). If TimeSpanStyles.AssumeNegative is not used, `format` must include a literal negative sign symbol (such as "\-") to successfully parse a negative time interval.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.

```
C#

public static TimeSpan ParseExact (ReadOnlySpan<char> input, string[]
formats, IFormatProvider? formatProvider,
System.Globalization.TimeSpanStyles styles =
System.Globalization.TimeSpanStyles.None);
```

Parameters

`input` ReadOnlySpan<Char>

A span that specifies the time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`styles` TimeSpanStyles

A bitwise combination of enumeration values that defines the style elements that may be present in input.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `formats`, `formatProvider`, and `styles`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

ParseExact(String, String[], IFormatProvider, TimeSpanStyles)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information, and styles. The format of the string representation must match one of the specified formats exactly.

```
C#

public static TimeSpan ParseExact (string input, string[] formats,
IFormatProvider? formatProvider, System.Globalization.TimeSpanStyles
styles);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`styles` TimeSpanStyles

A bitwise combination of enumeration values that defines the style elements that may be present in input.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `formats`, `formatProvider`, and `styles`.

Exceptions

ArgumentException

`styles` is an invalid TimeSpanStyles value.

ArgumentNullException

`input` is `null`.

FormatException

`input` has an invalid format.

OverflowException

`input` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components in `input` is outside its valid range.

Examples

The following example calls the ParseExact(String, String[], IFormatProvider, TimeSpanStyles) method to convert each element of a string array to a TimeSpan value. The strings can represent a time interval in either the general short format or the general long format.

In addition, the example changes the way in which the time interval parsing methods interpret a single digit. Ordinarily, a single digit is interpreted as the number of days in a time interval. Instead, the `%h` custom format string is used to interpret a single digit as the number of hours. For this change to be effective, note that the `%h` custom format string must precede the other format strings in the `formats` array. Also note from the output that the TimeSpanStyles.AssumeNegative flag specified in the method call is used only when parsing a string with this format specifier.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
        string[] inputs = { "3", "16:42", "1:6:52:35.0625",
                            "1:6:52:35,0625" };
        string[] formats = { "%h", "g", "G" };
        TimeSpan interval;
        CultureInfo culture = new CultureInfo("de-DE");

        // Parse each string in inputs using formats and the de-DE culture.
        foreach (string input in inputs) {
           try {
              interval = TimeSpan.ParseExact(input, formats, culture,

 TimeSpanStyles.AssumeNegative);
              Console.WriteLine("{0} --> {1:c}", input, interval);
           }
           catch (FormatException) {
              Console.WriteLine("{0} --> Bad Format", input);
           }
           catch (OverflowException) {
              Console.WriteLine("{0} --> Overflow", input);
           }
        }
    }
 }
 // The example displays the following output:
 //       3 --> -03:00:00
 //       16:42 --> 16:42:00
 //       1:6:52:35.0625 --> Bad Format
 //       1:6:52:35,0625 --> 1.06:52:35.0625000
```

Remarks

The ParseExact(String, String[], IFormatProvider, TimeSpanStyles) method parses the string representation of a time interval, which must be in one of the formats defined by the `formats` parameter, except that leading and trailing white-space characters are ignored. Because `input` must exactly conform to one of the formats specified in `formats`, you should always use exception handling when converting a string input by the user to a time interval. If you prefer not to use exception handling, you can call the TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan) method instead.

The `formats` parameter is a string array whose elements consist of either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings. `input` must correspond exactly to a member of `formats` for the parse operation to succeed. The parse operation attempts to match `input` to each element in `formats` starting with the first element in the array.

> **Important**
>
> The **ParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if the format string used to parse `input` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if the format string used to parse `input` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

The `styles` parameter affects the interpretation of strings that are parsed using custom format strings. It determines whether `input` is interpreted as a negative time interval only if a negative sign is present (TimeSpanStyles.None), or whether it is always interpreted as a negative time interval (TimeSpanStyles.AssumeNegative). If TimeSpanStyles.AssumeNegative is not used, `format` must include a literal negative sign symbol (such as "\-") to successfully parse a negative time interval.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(String, String[], IFormatProvider)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified array of format strings and culture-specific format information. The format of the string representation must match one of the specified formats exactly.

C#

```
public static TimeSpan ParseExact (string input, string[] formats,
IFormatProvider? formatProvider);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formats` String[]

An array of standard or custom format strings that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `formats` and `formatProvider`.

Exceptions

ArgumentNullException

`input` is `null`.

FormatException

`input` has an invalid format.

OverflowException

`input` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components in `input` is outside its valid range.

Examples

The following example calls the ParseExact(String, String[], IFormatProvider) method to convert each element of a string array to a TimeSpan value. The example interprets the strings using the formatting conventions of the French - France ("fr-FR") culture. The strings can represent a time interval in either the general short format or the general long format.

In addition, the example changes the way in which the time interval parsing methods interpret a single digit. Ordinarily, a single digit is interpreted as the number of days in a time interval. Instead, the `%h` custom format string is used to interpret a single digit as the number of hours. For this change to be effective, note that the `%h` custom format string must precede the other format strings in the `formats` array.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] inputs = { "3", "16:42", "1:6:52:35.0625",
```

```
                                "1:6:52:35,0625" };
        string[] formats = { "g", "G", "%h"};
        TimeSpan interval;
        CultureInfo culture = new CultureInfo("fr-FR");

        // Parse each string in inputs using formats and the fr-FR culture.
        foreach (string input in inputs) {
           try {
              interval = TimeSpan.ParseExact(input, formats, culture);
              Console.WriteLine("{0} --> {1:c}", input, interval);
           }
           catch (FormatException) {
              Console.WriteLine("{0} --> Bad Format", input);
           }
           catch (OverflowException) {
              Console.WriteLine("{0} --> Overflow", input);
           }
        }
     }
  }
  // The example displays the following output:
  //       3 --> 03:00:00
  //       16:42 --> 16:42:00
  //       1:6:52:35.0625 --> Bad Format
  //       1:6:52:35,0625 --> 1.06:52:35.0625000
```

Remarks

The ParseExact(String, String, IFormatProvider) method parses the string representation of a time interval, which must be in one of the formats defined by the `formats` parameter, except that leading and trailing white-space characters are ignored. Because `input` must exactly conform to one of the formats specified in `formats`, you should always use exception handling when converting a string input by the user to a time interval. If you prefer not to use exception handling, you can call the TryParseExact(String, String[], IFormatProvider, TimeSpan) method instead.

The `formats` parameter is a string array whose elements consist of either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings. `input` must correspond exactly to a member of `formats` for the parse operation to succeed. The parse operation attempts to match `input` to each element in `formats` starting with the first element in the array.

> **Important**

The **ParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if the format string used to parse `input` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if the format string used to parse `input` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

See also

- TryParseExact(String, String[], IFormatProvider, TimeSpan)
- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(String, String, IFormatProvider)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static TimeSpan ParseExact (string input, string format,
IFormatProvider? formatProvider);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`format` String

A standard or custom format string that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `format` and `formatProvider`.

Exceptions

ArgumentNullException

`input` is `null`.

FormatException

`input` has an invalid format.

OverflowException

`input` represents a number that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

-or-

At least one of the days, hours, minutes, or seconds components in `input` is outside its valid range.

Examples

The following example uses the ParseExact(String, String, IFormatProvider) method to parse several string representations of time intervals using various format strings and cultures.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string intervalString, format;
      TimeSpan interval;
      CultureInfo culture;

      // Parse hour:minute value with "g" specifier current culture.
      intervalString = "17:14";
      format = "g";
      culture = CultureInfo.CurrentCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'",
                           intervalString, format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse hour:minute:second value with "G" specifier.
      intervalString = "17:14:48";
```

```
      format = "G";
      culture = CultureInfo.InvariantCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse hours:minute.second value with "G" specifier
      // and current (en-US) culture.
      intervalString = "17:14:48.153";
      format = "G";
      culture = CultureInfo.CurrentCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with "G" specifier
      // and current (en-US) culture.
      intervalString = "3:17:14:48.153";
      format = "G";
      culture = CultureInfo.CurrentCulture;
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48.153";
      format = "G";
      culture = new CultureInfo("fr-FR");
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
```

```
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48,153";
      format = "G";
      try {
         interval = TimeSpan.ParseExact(intervalString, format, culture);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse a single number using the "c" standard format string.
      intervalString = "12";
      format = "c";
      try {
         interval = TimeSpan.ParseExact(intervalString, format, null);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }

      // Parse a single number using the "%h" custom format string.
      format = "%h";
      try {
         interval = TimeSpan.ParseExact(intervalString, format, null);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }
```

```
      // Parse a single number using the "%s" custom format string.
      format = "%s";
      try {
         interval = TimeSpan.ParseExact(intervalString, format, null);
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      }
      catch (FormatException) {
         Console.WriteLine("'{0}': Bad Format for '{1}'", intervalString,
format);
      }
      catch (OverflowException) {
         Console.WriteLine("'{0}': Overflow", intervalString);
      }
   }
}
// The example displays the following output:
//       '17:14' --> 17:14:00
//       '17:14:48': Bad Format for 'G'
//       '17:14:48.153': Bad Format for 'G'
//       '3:17:14:48.153' --> 3.17:14:48.1530000
//       '3:17:14:48.153': Bad Format for 'G'
//       '3:17:14:48,153' --> 3.17:14:48.1530000
//       '12' --> 12.00:00:00
//       '12' --> 12:00:00
//       '12' --> 00:00:12
```

Remarks

The ParseExact(String, String, IFormatProvider) method parses the string representation of a time interval, which must be in the format defined by the `format` parameter, except that leading and trailing white-space characters are ignored. Because `input` must conform to the format of `format` exactly, you should always use exception handling when converting a string input by the user to a time interval. If you prefer not to use exception handling, you can call the TryParseExact(String, String, IFormatProvider, TimeSpan) method instead.

The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.

> **ⓘ Important**
>
> The **ParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if `format` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use

the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if `format` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles)

Converts the char span of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static TimeSpan ParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, IFormatProvider? formatProvider,
System.Globalization.TimeSpanStyles styles =
System.Globalization.TimeSpanStyles.None);
```

Parameters

`input` ReadOnlySpan<Char>

A span that specifies the time interval to convert.

`format` ReadOnlySpan<Char>

A standard or custom format string that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`styles` TimeSpanStyles

A bitwise combination of enumeration values that defines the style elements that may be present in `input`.

Returns

TimeSpan

A time interval that corresponds to `input`, as specified by `format` and `formatProvider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Subtract(TimeSpan) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new TimeSpan object whose value is the difference between the specified TimeSpan object and this instance.

```
C#
public TimeSpan Subtract (TimeSpan ts);
```

Parameters

`ts` TimeSpan

The time interval to be subtracted.

Returns

TimeSpan

A new time interval whose value is the result of the value of this instance minus the value of `ts`.

Exceptions

OverflowException

The return value is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example uses the Subtract method to calculate the difference between a single TimeSpan value and each of the time intervals in an array. Note that, because TimeSpan format strings do not include negative signs in the result string, the example uses conditional logic to include a negative sign with negative time intervals.

```
C#
```

```
TimeSpan baseTimeSpan = new TimeSpan(1, 12, 15, 16);

// Create an array of timespan intervals.
TimeSpan[] intervals = {
   TimeSpan.FromDays(1.5),
   TimeSpan.FromHours(1.5),
   TimeSpan.FromMinutes(45),
   TimeSpan.FromMilliseconds(505),
   new TimeSpan(1, 17, 32, 20),
   new TimeSpan(-8, 30, 0)
};

// Calculate a new time interval by adding each element to the base
interval.
foreach (var interval in intervals)
   Console.WriteLine(@"{0,-10:g} - {3}{1,15:%d\:hh\:mm\:ss\.ffff} = {4}
{2:%d\:hh\:mm\:ss\.ffff}",
                     baseTimeSpan, interval,
baseTimeSpan.Subtract(interval),
                     interval < TimeSpan.Zero ? "-" : "",
                     baseTimeSpan < interval.Duration() ? "-" : "");

// The example displays the following output:
//       1:12:15:16 - 1:12:00:00.0000 = 0:00:15:16.0000
//       1:12:15:16 - 0:01:30:00.0000 = 1:10:45:16.0000
//       1:12:15:16 - 0:00:45:00.0000 = 1:11:30:16.0000
//       1:12:15:16 - 0:00:00:00.5050 = 1:12:15:15.4950
//       1:12:15:16 - 1:17:32:20.0000 = -0:05:17:04.0000
//       1:12:15:16 - -0:07:30:00.0000 = 1:19:45:16.0000
```

Remarks

The return value must be between TimeSpan.MinValue and TimeSpan.MaxValue; otherwise, an exception is thrown.

The return value is a new TimeSpan; the original TimeSpan is not modified.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current TimeSpan object to its equivalent string representation.

Overloads

Expand table

ToString(String, IFormatProvider)	Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format and culture-specific formatting information.
ToString(String)	Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format.
ToString()	Converts the value of the current TimeSpan object to its equivalent string representation.

ToString(String, IFormatProvider)

Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format and culture-specific formatting information.

C#

```
public string ToString (string? format, IFormatProvider? formatProvider);
```

Parameters

format String

A standard or custom TimeSpan format string.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the current TimeSpan value, as specified by `format` and `formatProvider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The `format` parameter is not recognized or is not supported.

Examples

The following example calls the ToString(String, IFormatProvider) method to format two time intervals. The example calls the method twice for each format string, first to display it using the conventions of the en-US culture and then to display it using the conventions of the fr-FR culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      TimeSpan[] intervals = { new TimeSpan(38, 30, 15),
                               new TimeSpan(16, 14, 30) };
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("fr-FR") };
      string[] formats = {"c", "g", "G", @"hh\:mm\:ss" };
      Console.WriteLine("{0,12}      Format  {1,22}  {2,22}\n",
                        "Interval", cultures[0].Name, cultures[1].Name);

      foreach (var interval in intervals) {
         foreach (var fmt in formats)
            Console.WriteLine("{0,12}  {1,10}  {2,22}  {3,22}",
```

```
                                interval, fmt,
                                interval.ToString(fmt, cultures[0]),
                                interval.ToString(fmt, cultures[1]));
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//        Interval      Format                    en-US
fr-FR
//
//      1.14:30:15           c              1.14:30:15
1.14:30:15
//      1.14:30:15           g              1:14:30:15
1:14:30:15
//      1.14:30:15           G      1:14:30:15.0000000
1:14:30:15,0000000
//      1.14:30:15  hh\:mm\:ss                14:30:15
14:30:15
//
//        16:14:30           c                16:14:30
16:14:30
//        16:14:30           g                16:14:30
16:14:30
//        16:14:30           G      0:16:14:30.0000000
0:16:14:30,0000000
//        16:14:30  hh\:mm\:ss                16:14:30
16:14:30
```

Remarks

The `format` parameter can be any valid standard or custom format specifier for TimeSpan values. If `format` is equal to String.Empty or is `null`, the return value of the current TimeSpan object is formatted with the common format specifier ("c"). If format is any other value, the method throws a FormatException.

> **Important**
>
> The custom format strings for **TimeSpan** values do not include a date or time separator. If you want to include these elements in your format string, you must treat them as character literals. See the example for an illustration, and see the **Custom TimeSpan Format Strings** topic for more information.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about format strings for TimeSpan values, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used. If `format` is a custom format string, the `formatProvider` parameter is ignored.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String)

Converts the value of the current TimeSpan object to its equivalent string representation by using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

format String

A standard or custom TimeSpan format string.

Returns

String

The string representation of the current TimeSpan value in the format specified by the `format` parameter.

Exceptions

FormatException

The `format` parameter is not recognized or is not supported.

Examples

The following example uses standard and custom TimeSpan format strings to display the string representation of each element in an array of TimeSpan values

```
C#

TimeSpan[] spans = {
   TimeSpan.Zero,
   new TimeSpan(-14, 0, 0, 0, 0),
   new TimeSpan(1, 2, 3),
   new TimeSpan(0, 0, 0, 0, 250),
   new TimeSpan(99, 23, 59, 59, 999),
   new TimeSpan(3, 0, 0),
   new TimeSpan(0, 0, 0, 0, 25)
};

string[] fmts = { "c", "g", "G", @"hh\:mm\:ss", "%m' min.'" };
foreach (TimeSpan span in spans)
```

```
{
   foreach (string fmt in fmts)
      Console.WriteLine("{0}: {1}", fmt, span.ToString(fmt));

   Console.WriteLine();
}
// The example displays the following output:
//       c: 00:00:00
//       g: 0:00:00
//       G: 0:00:00:00.0000000
//       hh\:mm\:ss: 00:00:00
//       %m' min.': 0 min.
//
//       c: -14.00:00:00
//       g: -14:0:00:00
//       G: -14:00:00:00.0000000
//       hh\:mm\:ss: 00:00:00
//       %m' min.': 0 min.
//
//       c: 01:02:03
//       g: 1:02:03
//       G: 0:01:02:03.0000000
//       hh\:mm\:ss: 01:02:03
//       %m' min.': 2 min.
//
//       c: 00:00:00.2500000
//       g: 0:00:00.25
//       G: 0:00:00:00.2500000
//       hh\:mm\:ss: 00:00:00
//       %m' min.': 0 min.
//
//       c: 99.23:59:59.9990000
//       g: 99:23:59:59.999
//       G: 99:23:59:59.9990000
//       hh\:mm\:ss: 23:59:59
//       %m' min.': 59 min.
//
//       c: 03:00:00
//       g: 3:00:00
//       G: 0:03:00:00.0000000
//       hh\:mm\:ss: 03:00:00
//       %m' min.': 0 min.
//
//       c: 00:00:00.0250000
//       g: 0:00:00.025
//       G: 0:00:00:00.0250000
//       hh\:mm\:ss: 00:00:00
//       %m' min.': 0 min.
```

Remarks

The `format` parameter can be any valid standard or custom format specifier for TimeSpan values. If `format` is equal to String.Empty or is `null`, the return value of the current TimeSpan object is formatted with the common format specifier ("c"). If `format` is any other value, the method throws a FormatException.

If `format` is a standard format string, the format of the returned string is defined by the formatting conventions of the current culture.

> **Important**
>
> The custom format strings for **TimeSpan** values do not include a date or time separator. If you want to include these elements in your format string, you must treat them as character literals. See the example for an illustration, and see the **Custom TimeSpan Format Strings** topic for more information.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about format strings for TimeSpan values, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString()

Converts the value of the current TimeSpan object to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the current TimeSpan value.

Examples

The following example displays the strings returned by calling the ToString method with a number of TimeSpan values. Note that although the example does not call the ToString method directly, it is called by the Console.WriteLine method when it attempts to convert a TimeSpan value to its string representation.

C#

```
TimeSpan span;

// Initialize a time span to zero.
span = TimeSpan.Zero;
Console.WriteLine(span);

// Initialize a time span to 14 days.
span = new TimeSpan(-14, 0, 0, 0, 0);
Console.WriteLine(span);

// Initialize a time span to 1:02:03.
span = new TimeSpan(1, 2, 3);
Console.WriteLine(span);

// Initialize a time span to 250 milliseconds.
span = new TimeSpan(0, 0, 0, 0, 250);
Console.WriteLine(span);
```

```
 // Initialize a time span to 99 days, 23 hours, 59 minutes, and 59.999
seconds.
 span = new TimeSpan(99, 23, 59, 59, 999);
 Console.WriteLine(span);

 // Initialize a time span to 3 hours.
 span = new TimeSpan(3, 0, 0);
 Console.WriteLine(span);

 // Initialize a timespan to 25 milliseconds.
 span = new TimeSpan(0, 0, 0, 0, 25);
 Console.WriteLine(span);

 // The example displays the following output:
 //       00:00:00
 //       -14.00:00:00
 //       01:02:03
 //       00:00:00.2500000
 //       99.23:59:59.9990000
 //       03:00:00
 //       00:00:00.0250000
```

Remarks

The returned string is formatted with the "c" format specifier and has the following format:

[-][*d*.]*hh*:*mm*:*ss*[.*fffffff*]

Elements in square brackets ([and]) may not be included in the returned string. Colons and periods (: and.) are literal characters. The non-literal elements are listed in the following table. Note that the string returned by the ToString() method is not culture-sensitive.

Expand table

Item	Description
"-"	A minus sign, which indicates a negative time interval. No sign is included for a positive time span.
"*d*"	The number of days in the time interval. This element is omitted if the time interval is less than one day.
"*hh*"	The number of hours in the time interval, ranging from 0 to 23.
"*mm*"	The number of minutes in the time interval, ranging from 0 to 59.
"*ss*"	The number of seconds in the time interval, ranging from 0 to 59.

Item	Description
"*fffffff*"	Fractional seconds in the time interval. This element is omitted if the time interval does not include fractional seconds. If present, fractional seconds are always expressed using seven decimal digits.

Notes to Callers

Support for formatting TimeSpan values was added in the .NET Framework 4. However, the ToString() method overload remains culture-insensitive. Its behavior remains unchanged from previous versions of the .NET Framework. To control the formatting of a TimeSpan value, call the ToString(String) or ToString(String, IFormatProvider) overload.

See also

- String
- Parse(String)
- TryParse(String, TimeSpan)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

TimeSpan.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current timespan number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? formatProvider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`formatProvider` IFormatProvider

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current timespan number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? formatProvider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the charactes that represent a standard or custom format string that defines the acceptable format for `destination`.

`formatProvider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified string representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.

Overloads

Expand table

TryParse(String, IFormatProvider, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeSpan)	Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, TimeSpan)	Converts the span representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.
TryParse(String, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.

TryParse(String, IFormatProvider, TimeSpan)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse (string? input, IFormatProvider?
formatProvider, out TimeSpan result);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`. This operation returns `false` if the `input` parameter is `null` or Empty, has an invalid format, represents a time interval that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue, or has at least one days, hours, minutes, or seconds component outside its valid range.

Examples

The following example defines an array of CultureInfo objects, and uses each object in calls to the TryParse(String, IFormatProvider, TimeSpan) method to parse the elements in a string array. The example illustrates how the conventions of a specific culture influence the formatting operation.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
      string[] values = { "6", "6:12", "6:12:14", "6:12:14:45",
                          "6.12:14:45", "6:12:14:45.3448",
                          "6:12:14:45,3448", "6:34:14:45" };
      CultureInfo[] cultures = { new CultureInfo("en-US"),
                                 new CultureInfo("ru-RU"),
                                 CultureInfo.InvariantCulture };

      string header = String.Format("{0,-17}", "String");
      foreach (CultureInfo culture in cultures)
         header += culture.Equals(CultureInfo.InvariantCulture) ?
                      String.Format("{0,20}", "Invariant") :
                      String.Format("{0,20}", culture.Name);

      Console.WriteLine(header);
      Console.WriteLine();

      foreach (string value in values)
      {
         Console.Write("{0,-17}", value);
         foreach (CultureInfo culture in cultures)
         {
            TimeSpan interval = new TimeSpan();
            if (TimeSpan.TryParse(value, culture, out interval))
               Console.Write("{0,20}", interval.ToString("c"));
            else
               Console.Write("{0,20}", "Unable to Parse");
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    String                          en-US               ru-RU
Invariant
//
//    6                         6.00:00:00          6.00:00:00
6.00:00:00
//    6:12                        06:12:00            06:12:00
06:12:00
//    6:12:14                     06:12:14            06:12:14
06:12:14
//    6:12:14:45                6.12:14:45          6.12:14:45
6.12:14:45
//    6.12:14:45                6.12:14:45          6.12:14:45
6.12:14:45
//    6:12:14:45.3448    6.12:14:45.3448000     Unable to Parse
6.12:14:45.3448000
//    6:12:14:45,3448       Unable to Parse  6.12:14:45.3448000
Unable to Parse
//    6:34:14:45            Unable to Parse     Unable to Parse
Unable to Parse
```

Remarks

For more information about this API, see Supplemental API remarks for TimeSpan.TryParse.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, TimeSpan)

Converts the span representation of a time interval to its TimeSpan equivalent by using the specified culture-specific formatting information, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> input, IFormatProvider?
formatProvider, out TimeSpan result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters representing the time interval to convert.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed

uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`. This operation returns `false` if the `input` parameter is `null` or Empty, has an invalid format, represents a time interval that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue, or has at least one days, hours, minutes, or seconds component outside its valid range.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(ReadOnlySpan<Char>, TimeSpan)

Converts the span representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, out TimeSpan result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the time interval to convert.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `s`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`. This operation returns `false` if the `s` parameter is `null` or Empty, has an invalid format, represents a time interval that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue, or has at least one days, hours, minutes, or seconds component outside its valid range.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, TimeSpan)

Converts the string representation of a time interval to its TimeSpan equivalent and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, out TimeSpan result);
```

Parameters

`s` String

A string that specifies the time interval to convert.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `s`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`. This operation returns `false` if the `s` parameter is `null` or Empty, has an invalid format, represents a time interval

that is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue, or has at least one days, hours, minutes, or seconds component outside its valid range.

Examples

The following example uses the TryParse method to create TimeSpan objects from valid TimeSpan strings and to indicate when the parse operation has failed because the time span string is invalid.

C#

```
using System;

public class TryParse
{
   private static void ParseTimeSpan(string intervalStr)
   {
      // Write the first part of the output line.
      Console.Write( "{0,20}   ", intervalStr );

      // Parse the parameter, and then convert it back to a string.
      TimeSpan intervalVal;
      if (TimeSpan.TryParse(intervalStr, out intervalVal))
      {
         string intervalToStr = intervalVal.ToString();

         // Pad the end of the TimeSpan string with spaces if it
         // does not contain milliseconds.
         int pIndex = intervalToStr.IndexOf(':');
         pIndex = intervalToStr.IndexOf('.', pIndex);
         if (pIndex < 0)
            intervalToStr += "        ";

         Console.WriteLine("{0,21}", intervalToStr);
         // Handle failure of TryParse method.
      }
      else
      {
         Console.WriteLine("Parse operation failed.");
      }
   }

   public static void Main()
   {
        Console.WriteLine( "{0,20}   {1,21}",
            "String to Parse", "TimeSpan" );
        Console.WriteLine( "{0,20}   {1,21}",
            "---------------", "---------------------" );

        ParseTimeSpan("0");
        ParseTimeSpan("14");
```

```
        ParseTimeSpan("1:2:3");
        ParseTimeSpan("0:0:0.250");
        ParseTimeSpan("10.20:30:40.50");
        ParseTimeSpan("99.23:59:59.9999999");
        ParseTimeSpan("0023:0059:0059.0099");
        ParseTimeSpan("23:0:0");
        ParseTimeSpan("24:0:0");
        ParseTimeSpan("0:59:0");
        ParseTimeSpan("0:60:0");
        ParseTimeSpan("0:0:59");
        ParseTimeSpan("0:0:60");
        ParseTimeSpan("10:");
        ParseTimeSpan("10:0");
        ParseTimeSpan(":10");
        ParseTimeSpan("0:10");
        ParseTimeSpan("10:20:");
        ParseTimeSpan("10:20:0");
        ParseTimeSpan(".123");
        ParseTimeSpan("0.12:00");
        ParseTimeSpan("10.");
        ParseTimeSpan("10.12");
        ParseTimeSpan("10.12:00");
    }
}
//            String to Parse                TimeSpan
//            ---------------   ---------------------
//                          0        00:00:00
//                         14     14.00:00:00
//                      1:2:3        01:02:03
//                  0:0:0.250        00:00:00.2500000
//             10.20:30:40.50     10.20:30:40.5000000
//        99.23:59:59.9999999     99.23:59:59.9999999
//        0023:0059:0059.0099        23:59:59.0099000
//                     23:0:0        23:00:00
//                     24:0:0   Parse operation failed.
//                     0:59:0        00:59:00
//                     0:60:0   Parse operation failed.
//                     0:0:59        00:00:59
//                     0:0:60   Parse operation failed.
//                        10:   Parse operation failed.
//                       10:0        10:00:00
//                        :10   Parse operation failed.
//                       0:10        00:10:00
//                     10:20:   Parse operation failed.
//                    10:20:0        10:20:00
//                       .123   Parse operation failed.
//                    0.12:00        12:00:00
//                        10.   Parse operation failed.
//                      10.12   Parse operation failed.
//                   10.12:00     10.12:00:00
```

Remarks

For more information about this API, see Supplemental API remarks for TimeSpan.TryParse.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.TryParseExact Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a time interval to its TimeSpan equivalent, and returns a value that indicates whether the conversion succeeded. The format of the string representation must match a specified format exactly.

Overloads

Expand table

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles, and returns a value that indicates whether the conversion succeeded. The format of the string representation must match the specified format exactly.
TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information and styles. The format of the string representation must match the specified format exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.
TryParseExact(String, String[], IFormatProvider, TimeSpan)	Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.

TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan)	Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.
TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.
TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpan)	Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.
TryParseExact(String, String, IFormatProvider, TimeSpan)	Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpanStyles, TimeSpan)

Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information, and styles, and returns a value that indicates whether the conversion succeeded. The format of the string representation must match the specified format exactly.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, IFormatProvider? formatProvider,
System.Globalization.TimeSpanStyles styles, out TimeSpan result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a time interval to convert.

`format` ReadOnlySpan<Char>

A span containing the charactes that represent a standard or custom format string that defines the acceptable format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`styles` TimeSpanStyles

One or more enumeration values that indicate the style of `input`.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format, culture-specific format information and styles. The format of the string representation must match the specified format exactly.

```
C#

public static bool TryParseExact (string? input, string? format,
IFormatProvider? formatProvider, System.Globalization.TimeSpanStyles
styles, out TimeSpan result);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`format` String

A standard or custom format string that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`styles` TimeSpanStyles

One or more enumeration values that indicate the style of `input`.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Examples

The following example uses the ParseExact(String, String, IFormatProvider) method to parse several string representations of time intervals using various format strings and cultures. It also uses the TimeSpanStyles.AssumeNegative value to interpret each string as a negative time interval. The output from the example illustrates that the TimeSpanStyles.AssumeNegative style affects the return value only when it is used with custom format strings.

```
C#
```

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string intervalString, format;
      TimeSpan interval;
      CultureInfo culture = null;

      // Parse hour:minute value with custom format specifier.
      intervalString = "17:14";
      format = "h\\:mm";
      culture = CultureInfo.CurrentCulture;
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse hour:minute:second value with "g" specifier.
      intervalString = "17:14:48";
      format = "g";
      culture = CultureInfo.InvariantCulture;
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse hours:minute.second value with custom format specifier.
      intervalString = "17:14:48.153";
      format = @"h\:mm\:ss\.fff";
      culture = null;
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse days:hours:minute.second value with "G" specifier
      // and current (en-US) culture.
      intervalString = "3:17:14:48.153";
      format = "G";
```

```
      culture = CultureInfo.CurrentCulture;
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse days:hours:minute.second value with a custom format
specifier.
      intervalString = "3:17:14:48.153";
      format = @"d\:hh\:mm\:ss\.fff";
      culture = null;
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48,153";
      format = "G";
      culture = new CultureInfo("fr-FR");
      if (TimeSpan.TryParseExact(intervalString, format,
                                 culture, TimeSpanStyles.AssumeNegative,
out interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse a single number using the "c" standard format string.
      intervalString = "12";
      format = "c";
      if (TimeSpan.TryParseExact(intervalString, format,
                                 null, TimeSpanStyles.AssumeNegative, out
interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse a single number using the "%h" custom format string.
      format = "%h";
      if (TimeSpan.TryParseExact(intervalString, format,
                                 null, TimeSpanStyles.AssumeNegative, out
interval))
```

```
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);

      // Parse a single number using the "%s" custom format string.
      format = "%s";
      if (TimeSpan.TryParseExact(intervalString, format,
                                 null, TimeSpanStyles.AssumeNegative, out
interval))
         Console.WriteLine("'{0}' ({1}) --> {2}", intervalString, format,
interval);
      else
         Console.WriteLine("Unable to parse '{0}' using format {1}",
                           intervalString, format);
   }
}
// The example displays the following output:
//    '17:14' (h\:mm) --> -17:14:00
//    '17:14:48' (g) --> 17:14:48
//    '17:14:48.153' (h\:mm\:ss\.fff) --> -17:14:48.1530000
//    '3:17:14:48.153' (G) --> 3.17:14:48.1530000
//    '3:17:14:48.153' (d\:hh\:mm\:ss\.fff) --> -3.17:14:48.1530000
//    '3:17:14:48,153' (G) --> 3.17:14:48.1530000
//    '12' (c) --> 12.00:00:00
//    '12' (%h) --> -12:00:00
//    '12' (%s) --> -00:00:12
```

Remarks

The TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan) method parses the string representation of a time interval, which must be in the format defined by the `format` parameter, except that leading and trailing white-space characters are ignored. This method is similar to the ParseExact(String, String, IFormatProvider, TimeSpanStyles) method, except that it does not throw an exception if the conversion fails.

The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if `format` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

The `styles` parameter affects the interpretation of strings parsed using custom format strings. It determines whether `input` is interpreted as a negative time interval only if a negative sign is present (TimeSpanStyles.None), or whether it is always interpreted as a negative time interval (TimeSpanStyles.AssumeNegative). If TimeSpanStyles.AssumeNegative is not used, `format` must include a literal negative sign symbol (such as "\-") to successfully parse a negative time interval.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpanStyles, TimeSpan)

Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input, string?[]?
formats, IFormatProvider? formatProvider,
System.Globalization.TimeSpanStyles styles, out TimeSpan result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the acceptable formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`styles` TimeSpanStyles

One or more enumeration values that indicate the style of `input`.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String[], IFormatProvider, TimeSpan)

Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (string? input, string?[]? formats,
IFormatProvider? formatProvider, out TimeSpan result);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the acceptable formats of `input`.

`formatProvider` IFormatProvider

An object that provides culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParseExact(String, String[], IFormatProvider, TimeSpan) method to convert each element of a string array to a TimeSpan value. The example interprets the strings by using the formatting conventions of the French - France ("fr-FR") culture. The strings can represent a time interval in either the general short format or the general long format.

In addition, the example changes the way in which the time interval parsing methods interpret a single digit. Ordinarily, a single digit is interpreted as the number of days in a time interval. Instead, the `%h` custom format string is used to interpret a single digit as the number of hours. For this change to be effective, note that the `%h` custom format string must precede the other format strings in the `formats` array.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] inputs = { "3", "16:42", "1:6:52:35.0625",
                          "1:6:52:35,0625" };
      string[] formats = { "g", "G", "%h"};
      TimeSpan interval;
      CultureInfo culture = new CultureInfo("fr-FR");

      // Parse each string in inputs using formats and the fr-FR culture.
      foreach (string input in inputs) {
         if(TimeSpan.TryParseExact(input, formats, culture, out
interval))
            Console.WriteLine("{0} --> {1:c}", input, interval);
         else
            Console.WriteLine("Unable to parse {0}", input);
      }
   }
}
// The example displays the following output:
//       3 --> 03:00:00
//       16:42 --> 16:42:00
```

```
//       Unable to parse 1:6:52:35.0625
//       1:6:52:35,0625 --> 1.06:52:35.0625000
```

Remarks

The TryParseExact(String, String[], IFormatProvider, TimeSpan) method parses the string representation of a time interval, which must be in the format defined by one of the format strings specified by the `formats` parameter, except that leading and trailing white-space characters are ignored. This method is similar to the ParseExact(String, String[], IFormatProvider) method, except that it does not throw an exception if the conversion fails.

The `formats` parameter is a string array whose elements consist of either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings. `input` must correspond exactly to a member of `formats` for the parse operation to succeed. The parse operation attempts to match `input` to each element in `formats` starting with the first element in the array.

> **Important**
>
> The **TryParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if the format string used to parse `input` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if the format string used to parse `input` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.

- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan)

Converts the specified string representation of a time interval to its TimeSpan equivalent by using the specified formats, culture-specific format information and styles. The format of the string representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (string? input, string?[]? formats,
IFormatProvider? formatProvider, System.Globalization.TimeSpanStyles
styles, out TimeSpan result);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the acceptable formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`styles` TimeSpanStyles

One or more enumeration values that indicate the style of `input`.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan) method to convert each element of a string array to a TimeSpan value. The strings can represent a time interval in either the general short format or the general long format.

In addition, the example changes the way in which the time interval parsing methods interpret a single digit. Ordinarily, a single digit is interpreted as the number of days in a time interval. Instead, the `%h` custom format string is used to interpret a single digit as the number of hours. For this change to be effective, note that the `%h` custom format string must precede the other format strings in the `formats` array. Also note from the output that the TimeSpanStyles.AssumeNegative flag specified in the method call is used only when parsing a string with this format specifier.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] inputs = { "3", "16:42", "1:6:52:35.0625",
                          "1:6:52:35,0625" };
      string[] formats = { "%h", "g", "G" };
      TimeSpan interval;
      CultureInfo culture = new CultureInfo("fr-FR");

      // Parse each string in inputs using formats and the fr-FR culture.
      foreach (string input in inputs) {
         if(TimeSpan.TryParseExact(input, formats, culture,
                                   TimeSpanStyles.AssumeNegative, out
interval))
            Console.WriteLine("{0} --> {1:c}", input, interval);
         else
            Console.WriteLine("Unable to parse {0}", input);
      }
   }
}
// The example displays the following output:
//       3 --> -03:00:00
//       16:42 --> 16:42:00
//       Unable to parse 1:6:52:35.0625
//       1:6:52:35,0625 --> 1.06:52:35.0625000
```

Remarks

The TryParseExact(String, String[], IFormatProvider, TimeSpanStyles, TimeSpan) method parses the string representation of a time interval, which must be in the format defined by one of the format strings specified by the `formats` parameter, except that leading and trailing white-space characters are ignored. This method is similar to the ParseExact(String, String[], IFormatProvider, TimeSpanStyles) method, except that it does not throw an exception if the conversion fails.

The `formats` parameter is a string array whose elements consist of either a single standard format specifier, or one or more custom format specifiers that define the required format of `input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings. `input` must correspond exactly to a member of `formats` for the parse operation to succeed.

The parse operation attempts to match `input` to each element in `formats` starting with the first element in the array.

> **Important**
>
> The **ParseExact** method uses the conventions of the culture specified by the `formatProvider` parameter only if the format string used to parse `input` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if the format string used to parse `input` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

The `styles` parameter affects the interpretation of strings that are parsed using custom format strings. It determines whether `input` is interpreted as a negative time interval only if a negative sign is present (TimeSpanStyles.None), or whether it is always interpreted as a negative time interval (TimeSpanStyles.AssumeNegative). If TimeSpanStyles.AssumeNegative is not used, `format` must include a literal negative sign symbol (such as "\-") to successfully parse a negative time interval.

See also

- Standard TimeSpan Format Strings

- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParseExact(ReadOnlySpan<Char>, String[], IFormatProvider, TimeSpan)

Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified formats and culture-specific format information. The format of the string representation must match one of the specified formats exactly.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input, string?[]? formats, IFormatProvider? formatProvider, out TimeSpan result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a time interval to convert.

`formats` String[]

An array of standard or custom format strings that define the acceptable formats of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(ReadOnlySpan<Char>, ReadOnlySpan<Char>, IFormatProvider, TimeSpan)

Converts the specified span representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static bool TryParseExact (ReadOnlySpan<char> input,
ReadOnlySpan<char> format, IFormatProvider? formatProvider, out TimeSpan
result);
```

Parameters

`input` ReadOnlySpan<Char>

A span containing the characters that represent a time interval to convert.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParseExact(String, String, IFormatProvider, TimeSpan)

Converts the string representation of a time interval to its TimeSpan equivalent by using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly.

C#

```
public static bool TryParseExact (string? input, string? format,
IFormatProvider? formatProvider, out TimeSpan result);
```

Parameters

`input` String

A string that specifies the time interval to convert.

`format` String

A standard or custom format string that defines the required format of `input`.

`formatProvider` IFormatProvider

An object that supplies culture-specific formatting information.

`result` TimeSpan

When this method returns, contains an object that represents the time interval specified by `input`, or Zero if the conversion failed. This parameter is passed uninitialized.

Returns

Boolean

`true` if `input` was converted successfully; otherwise, `false`.

Examples

The following example uses the TryParseExact(String, String, IFormatProvider, TimeSpanStyles, TimeSpan) method to parse several string representations of time intervals using various format strings and cultures.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string intervalString, format;
      TimeSpan interval;
      CultureInfo culture;

      // Parse hour:minute value with "g" specifier current culture.
      intervalString = "17:14";
      format = "g";
      culture = CultureInfo.CurrentCulture;
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
```

```
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse hour:minute:second value with "G" specifier.
      intervalString = "17:14:48";
      format = "G";
      culture = CultureInfo.InvariantCulture;
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse hours:minute.second value with "G" specifier
      // and current (en-US) culture.
      intervalString = "17:14:48.153";
      format = "G";
      culture = CultureInfo.CurrentCulture;
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse days:hours:minute.second value with "G" specifier
      // and current (en-US) culture.
      intervalString = "3:17:14:48.153";
      format = "G";
      culture = CultureInfo.CurrentCulture;
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48.153";
      format = "G";
      culture = new CultureInfo("fr-FR");
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse days:hours:minute.second value with "G" specifier
      // and fr-FR culture.
      intervalString = "3:17:14:48,153";
      format = "G";
      if (TimeSpan.TryParseExact(intervalString, format, culture, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);
```

```
      // Parse a single number using the "c" standard format string.
      intervalString = "12";
      format = "c";
      if (TimeSpan.TryParseExact(intervalString, format, null, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse a single number using the "%h" custom format string.
      format = "%h";
      if (TimeSpan.TryParseExact(intervalString, format, null, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);

      // Parse a single number using the "%s" custom format string.
      format = "%s";
      if (TimeSpan.TryParseExact(intervalString, format, null, out
interval))
         Console.WriteLine("'{0}' --> {1}", intervalString, interval);
      else
         Console.WriteLine("Unable to parse {0}", intervalString);
   }
}
// The example displays the following output:
//       '17:14' --> 17:14:00
//       Unable to parse 17:14:48
//       Unable to parse 17:14:48.153
//       '3:17:14:48.153' --> 3.17:14:48.1530000
//       Unable to parse 3:17:14:48.153
//       '3:17:14:48,153' --> 3.17:14:48.1530000
//       '12' --> 12.00:00:00
//       '12' --> 12:00:00
//       '12' --> 00:00:12
```

Remarks

The TryParseExact(String, String, IFormatProvider, TimeSpan) method parses the string representation of a time interval, which must be in the format defined by the `format` parameter, except that leading and trailing white-space characters are ignored. This method is similar to the ParseExact(String, String, IFormatProvider) method, except that it does not throw an exception if the conversion fails.

The `format` parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of

`input`. For more information about valid format strings, see Standard TimeSpan Format Strings and Custom TimeSpan Format Strings.

> ⓘ **Important**
>
> The **TryParseExact(String, String, IFormatProvider, TimeSpan)** method uses the conventions of the culture specified by the `formatProvider` parameter only if `format` is a standard **TimeSpan** format string whose value is either "g" or "G". The "c", "t", and "T" standard format strings use the formatting conventions of the invariant culture. Custom format strings define the precise format of the input string and use literal characters to separate the components of a time interval.

The `formatProvider` parameter is an IFormatProvider implementation that provides culture-specific information about the format of the returned string if `format` is a standard format string. The `formatProvider` parameter can be any of the following:

- A CultureInfo object that represents the culture whose formatting conventions are to be reflected in the returned string. The DateTimeFormatInfo object returned by the CultureInfo.DateTimeFormat property defines the formatting of the returned string.
- A DateTimeFormatInfo object that defines the formatting of the returned string.
- A custom object that implements the IFormatProvider interface. Its IFormatProvider.GetFormat method returns a DateTimeFormatInfo object that provides formatting information.

If `formatProvider` is `null`, the DateTimeFormatInfo object that is associated with the current culture is used.

See also

- Standard TimeSpan Format Strings
- Custom TimeSpan Format Strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Addition(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two specified TimeSpan instances.

C#

```
public static TimeSpan operator + (TimeSpan t1, TimeSpan t2);
```

Parameters

`t1` TimeSpan

The first time interval to add.

`t2` TimeSpan

The second time interval to add.

Returns

TimeSpan

An object whose value is the sum of the values of `t1` and `t2`.

Exceptions

OverflowException

The resulting TimeSpan is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Remarks

The Addition method defines the addition operator for TimeSpan values. It enables code such as the following:

C#

```
TimeSpan time1 = new TimeSpan(1, 0, 0, 0);   // TimeSpan equivalent to 1
day.
TimeSpan time2 = new TimeSpan(12, 0, 0);     // TimeSpan equivalent to 1/2
day.
TimeSpan time3 = time1 + time2;              // Add the two time spans.

Console.WriteLine("  {0,12}\n +  {1,10}\n   {3}\n    {2,10}",
                  time1, time2, time3, new String('_', 10));

// The example displays the following output:
//           1.00:00:00
//        +    12:00:00
//          __________
//           1.12:00:00
```

Languages that do not support custom operators can call the Add method instead.

The equivalent method for this operator is TimeSpan.Add(TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(TimeSpan)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeSpan.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Division(TimeSpan, Double)	Returns a new TimeSpan object whose value is the result of dividing the specified `timeSpan` by the specified `divisor`.
Division(TimeSpan, TimeSpan)	Returns a new Double value that's the result of dividing `t1` by `t2`.

Division(TimeSpan, Double)

Returns a new TimeSpan object whose value is the result of dividing the specified `timeSpan` by the specified `divisor`.

C#

```
public static TimeSpan operator / (TimeSpan timeSpan, double divisor);
```

Parameters

`timeSpan` TimeSpan

Dividend or the value to be divided.

`divisor` Double

The value to be divided by.

Returns

TimeSpan

A new object that represents the value of `timeSpan` divided by the value of `divisor`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Division(TimeSpan, TimeSpan)

Returns a new Double value that's the result of dividing `t1` by `t2`.

C#

```
public static double operator / (TimeSpan t1, TimeSpan t2);
```

Parameters

`t1` TimeSpan

The divident or the value to be divided.

`t2` TimeSpan

The value to be divided by.

Returns

Double

A new value that represents result of dividing `t1` by the value of `t2`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Equality(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two TimeSpan instances are equal.

```
C#
public static bool operator == (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the values of `t1` and `t2` are equal; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the Equality operator.

```
C#
// Example of the TimeSpan relational operators.
using System;

class TSRelationalOpsDemo
```

```
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
```

```
         Right: TimeSpan( 2, 0, 1 )    02:00:01
                      Left == Right    False
                      Left >  Right    False
                      Left >= Right    False
                      Left != Right    True
                      Left <  Right    True
                      Left <= Right    True

        Right: TimeSpan( 2, 0, -1 )    01:59:59
                      Left == Right    False
                      Left >  Right    True
                      Left >= Right    True
                      Left != Right    True
                      Left <  Right    False
                      Left <= Right    False

 Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
 */
```

Remarks

The equivalent method for this operator is TimeSpan.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.GreaterThan(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified TimeSpan is greater than another specified TimeSpan.

C#

```
public static bool operator > (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the value of `t1` is greater than the value of `t2`; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the GreaterThan operator.

C#

```
// Example of the TimeSpan relational operators.
using System;

class TSRelationalOpsDemo
```

```
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
```

```
        Right: TimeSpan( 2, 0, 1 )    02:00:01
                     Left == Right    False
                     Left >  Right    False
                     Left >= Right    False
                     Left != Right    True
                     Left <  Right    True
                     Left <= Right    True

       Right: TimeSpan( 2, 0, -1 )    01:59:59
                     Left == Right    False
                     Left >  Right    True
                     Left >= Right    True
                     Left != Right    True
                     Left <  Right    False
                     Left <= Right    False

Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                     Left == Right    True
                     Left >  Right    False
                     Left >= Right    True
                     Left != Right    False
                     Left <  Right    False
                     Left <= Right    True
*/
```

Remarks

The equivalent method for this operator is TimeSpan.Compare(TimeSpan, TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.GreaterThanOrEqual(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified TimeSpan is greater than or equal to another specified TimeSpan.

```
C#

public static bool operator >= (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the value of `t1` is greater than or equal to the value of `t2`; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the GreaterThanOrEqual operator.

```
C#

// Example of the TimeSpan relational operators.
using System;
```

```
class TSRelationalOpsDemo
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                     Left == Right    True
                     Left >  Right    False
                     Left >= Right    True
                     Left != Right    False
                     Left <  Right    False
```

```
                      Left <= Right    True

         Right: TimeSpan( 2, 0, 1 )    02:00:01
                      Left == Right    False
                      Left >  Right    False
                      Left >= Right    False
                      Left != Right    True
                      Left <  Right    True
                      Left <= Right    True

        Right: TimeSpan( 2, 0, -1 )    01:59:59
                      Left == Right    False
                      Left >  Right    True
                      Left >= Right    True
                      Left != Right    True
                      Left <  Right    False
                      Left <= Right    False

 Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
 */
```

Remarks

The equivalent method for this operator is TimeSpan.Compare(TimeSpan, TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.Inequality(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two TimeSpan instances are not equal.

```
C#
public static bool operator != (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the values of `t1` and `t2` are not equal; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the Inequality operator.

```
C#
// Example of the TimeSpan relational operators.
using System;

class TSRelationalOpsDemo
```

```
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
```

```
        Right: TimeSpan( 2, 0, 1 )    02:00:01
                     Left == Right    False
                     Left >  Right    False
                     Left >= Right    False
                     Left != Right    True
                     Left <  Right    True
                     Left <= Right    True

       Right: TimeSpan( 2, 0, -1 )    01:59:59
                     Left == Right    False
                     Left >  Right    True
                     Left >= Right    True
                     Left != Right    True
                     Left <  Right    False
                     Left <= Right    False

Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                     Left == Right    True
                     Left >  Right    False
                     Left >= Right    True
                     Left != Right    False
                     Left <  Right    False
                     Left <= Right    True
*/
```

Remarks

The equivalent method for this operator is TimeSpan.Compare(TimeSpan, TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.LessThan(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified TimeSpan is less than another specified TimeSpan.

C#

```
public static bool operator < (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the value of `t1` is less than the value of `t2`; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the LessThan operator.

C#

```
// Example of the TimeSpan relational operators.
using System;

class TSRelationalOpsDemo
```

```
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
```

```
        Right: TimeSpan( 2, 0, 1 )    02:00:01
                     Left == Right    False
                     Left >  Right    False
                     Left >= Right    False
                     Left != Right    True
                     Left <  Right    True
                     Left <= Right    True

       Right: TimeSpan( 2, 0, -1 )    01:59:59
                     Left == Right    False
                     Left >  Right    True
                     Left >= Right    True
                     Left != Right    True
                     Left <  Right    False
                     Left <= Right    False

Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                     Left == Right    True
                     Left >  Right    False
                     Left >= Right    True
                     Left != Right    False
                     Left <  Right    False
                     Left <= Right    True
*/
```

Remarks

The equivalent method for this operator is TimeSpan.Compare(TimeSpan, TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.LessThanOrEqual(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified TimeSpan is less than or equal to another specified TimeSpan.

C#

```
public static bool operator <= (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The first time interval to compare.

t2 TimeSpan

The second time interval to compare.

Returns

Boolean

`true` if the value of `t1` is less than or equal to the value of `t2`; otherwise, `false`.

Examples

The following example compares several TimeSpan objects to a reference TimeSpan using the LessThanOrEqual operator.

C#

```
// Example of the TimeSpan relational operators.
using System;
```

```
class TSRelationalOpsDemo
{
    const string dataFmt = "{0,34}    {1}" ;

    // Compare TimeSpan parameters, and display them with the results.
    static void CompareTimeSpans( TimeSpan Left, TimeSpan Right,
        string RightText )
    {
        Console.WriteLine( );
        Console.WriteLine( dataFmt, "Right: " + RightText, Right );
        Console.WriteLine( dataFmt, "Left == Right", Left == Right );
        Console.WriteLine( dataFmt, "Left >  Right", Left > Right );
        Console.WriteLine( dataFmt, "Left >= Right", Left >= Right );
        Console.WriteLine( dataFmt, "Left != Right", Left != Right );
        Console.WriteLine( dataFmt, "Left <  Right", Left < Right );
        Console.WriteLine( dataFmt, "Left <= Right", Left <= Right );
    }

    static void Main( )
    {
        TimeSpan Left = new TimeSpan( 2, 0, 0 );

        Console.WriteLine(
            "This example of the TimeSpan relational operators " +
            "generates \nthe following output. It creates several " +
            "different TimeSpan \nobjects and compares them with " +
            "a 2-hour TimeSpan.\n" );
        Console.WriteLine( dataFmt,
            "Left: TimeSpan( 2, 0, 0 )", Left );

        // Create objects to compare with a 2-hour TimeSpan.
        CompareTimeSpans( Left, new TimeSpan( 0, 120, 0 ),
            "TimeSpan( 0, 120, 0 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, 1 ),
            "TimeSpan( 2, 0, 1 )" );
        CompareTimeSpans( Left, new TimeSpan( 2, 0, -1 ),
            "TimeSpan( 2, 0, -1 )" );
        CompareTimeSpans( Left, TimeSpan.FromDays( 1.0 / 12D ),
            "TimeSpan.FromDays( 1 / 12 )" );
    }
}

/*
This example of the TimeSpan relational operators generates
the following output. It creates several different TimeSpan
objects and compares them with a 2-hour TimeSpan.

         Left: TimeSpan( 2, 0, 0 )    02:00:00

      Right: TimeSpan( 0, 120, 0 )    02:00:00
                     Left == Right    True
                     Left >  Right    False
                     Left >= Right    True
                     Left != Right    False
                     Left <  Right    False
```

```
                      Left <= Right    True

         Right: TimeSpan( 2, 0, 1 )    02:00:01
                      Left == Right    False
                      Left >  Right    False
                      Left >= Right    False
                      Left != Right    True
                      Left <  Right    True
                      Left <= Right    True

        Right: TimeSpan( 2, 0, -1 )    01:59:59
                      Left == Right    False
                      Left >  Right    True
                      Left >= Right    True
                      Left != Right    True
                      Left <  Right    False
                      Left <= Right    False

 Right: TimeSpan.FromDays( 1 / 12 )    02:00:00
                      Left == Right    True
                      Left >  Right    False
                      Left >= Right    True
                      Left != Right    False
                      Left <  Right    False
                      Left <= Right    True
 */
```

Remarks

The equivalent method for this operator is TimeSpan.Compare(TimeSpan, TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Compare(TimeSpan, TimeSpan)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Multiply(Double, TimeSpan)	Returns a new TimeSpan object whose value is the result of multiplying the specified `factor` and the specified `timeSpan` instance.
Multiply(TimeSpan, Double)	Returns a new TimeSpan object whose value is the result of multiplying the specified `timeSpan` instance and the specified `factor`.

Multiply(Double, TimeSpan)

Returns a new TimeSpan object whose value is the result of multiplying the specified `factor` and the specified `timeSpan` instance.

C#

```
public static TimeSpan operator * (double factor, TimeSpan timeSpan);
```

Parameters

`factor` Double

The value to be multiplied by.

`timeSpan` TimeSpan

The value to be multiplied.

Returns

TimeSpan

A new object that represents the value of the specified `factor` multiplied by the value of the specified `timeSpan` instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Multiply(TimeSpan, Double)

Returns a new TimeSpan object whose value is the result of multiplying the specified `timeSpan` instance and the specified `factor`.

C#

```
public static TimeSpan operator * (TimeSpan timeSpan, double factor);
```

Parameters

`timeSpan` TimeSpan

The value to be multiplied.

`factor` Double

The value to be multiplied by.

Returns

TimeSpan

A new object that represents the value of the specified `timeSpan` instance multiplied by the value of the specified `factor`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.Subtraction(TimeSpan, TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts a specified TimeSpan from another specified TimeSpan.

C#

```
public static TimeSpan operator - (TimeSpan t1, TimeSpan t2);
```

Parameters

t1 TimeSpan

The minuend.

t2 TimeSpan

The subtrahend.

Returns

TimeSpan

An object whose value is the result of the value of `t1` minus the value of `t2`.

Exceptions

OverflowException

The return value is less than TimeSpan.MinValue or greater than TimeSpan.MaxValue.

Examples

The following example uses the TimeSpan subtraction operator to calculate the total length of the weekly work day. It also uses the TimeSpan addition operator to compute

the total time of the daily breaks before using it in a subtraction operation to compute the total actual daily working time.

C#

```
var startWork = new TimeSpan(08,00,00);
var endWork = new TimeSpan(18,30,00);
var lunchBreak = new TimeSpan(1, 0, 0);
var breaks = new TimeSpan(0, 30, 0);

Console.WriteLine("Length of work day: {0}",
                  endWork - startWork);
Console.WriteLine("Actual time worked: {0}",
                  endWork - startWork - (lunchBreak + breaks));

// The example displays the following output:
//     Length of work day: 10:30:00
//     Actual time worked: 09:00:00
```

Remarks

The equivalent method for this operator is TimeSpan.Subtract(TimeSpan)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeSpan.UnaryNegation(TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a TimeSpan whose value is the negated value of the specified instance.

C#

```
public static TimeSpan operator - (TimeSpan t);
```

Parameters

`t` TimeSpan

The time interval to be negated.

Returns

TimeSpan

An object that has the same numeric value as this instance, but the opposite sign.

Exceptions

OverflowException

The negated value of this instance cannot be represented by a TimeSpan; that is, the value of this instance is TimeSpan.MinValue.

Examples

The following example applies the UnaryNegation operator to several TimeSpan objects.

C#

```
// Example of the TimeSpan.Duration( ) and TimeSpan.Negate( ) methods,
// and the TimeSpan Unary Negation and Unary Plus operators.
```

```
using System;

class DuraNegaUnaryDemo
{
    const string dataFmt = "{0,22}{1,22}{2,22}" ;

    static void ShowDurationNegate( TimeSpan interval )
    {
        // Display the TimeSpan value and the results of the
        // Duration and Negate methods.
        Console.WriteLine( dataFmt,
            interval, interval.Duration( ), interval.Negate( ) );
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of TimeSpan.Duration( ), " +
            "TimeSpan.Negate( ), \nand the TimeSpan Unary " +
            "Negation and Unary Plus operators \n" +
            "generates the following output.\n" );
        Console.WriteLine( dataFmt,
            "TimeSpan", "Duration( )", "Negate( )" );
        Console.WriteLine( dataFmt,
            "--------", "-----------", "---------" );

        // Create TimeSpan objects and apply the Unary Negation
        // and Unary Plus operators to them.
        ShowDurationNegate( new TimeSpan( 1 ) );
        ShowDurationNegate( new TimeSpan( -1234567 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, 0, 10, -20, -30 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, -10, 20, -30, 40 ) );
        ShowDurationNegate(
            - new TimeSpan( 1, 10, 20, 40, 160 ) );
        ShowDurationNegate(
            - new TimeSpan( -10, -20, -30, -40, -50 ) );
    }
}

/*
This example of TimeSpan.Duration( ), TimeSpan.Negate( ),
and the TimeSpan Unary Negation and Unary Plus operators
generates the following output.

              TimeSpan           Duration( )             Negate( )
              --------           -----------             ---------
      00:00:00.0000001      00:00:00.0000001     -00:00:00.0000001
     -00:00:00.1234567      00:00:00.1234567      00:00:00.1234567
      00:09:39.9700000      00:09:39.9700000     -00:09:39.9700000
     -09:40:29.9600000      09:40:29.9600000      09:40:29.9600000
   -1.10:20:40.1600000    1.10:20:40.1600000    1.10:20:40.1600000
   10.20:30:40.0500000   10.20:30:40.0500000  -10.20:30:40.0500000
*/
```

Remarks

The equivalent method for this operator is TimeSpan.Negate()

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeSpan.UnaryPlus(TimeSpan) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the specified instance of TimeSpan.

C#

```
public static TimeSpan operator + (TimeSpan t);
```

Parameters

`t` TimeSpan

The time interval to return.

Returns

TimeSpan

The time interval specified by `t`.

Examples

The following example applies the UnaryPlus operator to several TimeSpan objects.

C#

```
// Example of the TimeSpan.Duration( ) and TimeSpan.Negate( ) methods,
// and the TimeSpan Unary Negation and Unary Plus operators.
using System;

class DuraNegaUnaryDemo
{
    const string dataFmt = "{0,22}{1,22}{2,22}" ;

    static void ShowDurationNegate( TimeSpan interval )
    {
```

```
        // Display the TimeSpan value and the results of the
        // Duration and Negate methods.
        Console.WriteLine( dataFmt,
            interval, interval.Duration( ), interval.Negate( ) );
    }

    static void Main()
    {
        Console.WriteLine(
            "This example of TimeSpan.Duration( ), " +
            "TimeSpan.Negate( ), \nand the TimeSpan Unary " +
            "Negation and Unary Plus operators \n" +
            "generates the following output.\n" );
        Console.WriteLine( dataFmt,
            "TimeSpan", "Duration( )", "Negate( )" );
        Console.WriteLine( dataFmt,
            "--------", "-----------", "---------" );

        // Create TimeSpan objects and apply the Unary Negation
        // and Unary Plus operators to them.
        ShowDurationNegate( new TimeSpan( 1 ) );
        ShowDurationNegate( new TimeSpan( -1234567 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, 0, 10, -20, -30 ) );
        ShowDurationNegate(
            + new TimeSpan( 0, -10, 20, -30, 40 ) );
        ShowDurationNegate(
            - new TimeSpan( 1, 10, 20, 40, 160 ) );
        ShowDurationNegate(
            - new TimeSpan( -10, -20, -30, -40, -50 ) );
    }
}

/*
This example of TimeSpan.Duration( ), TimeSpan.Negate( ),
and the TimeSpan Unary Negation and Unary Plus operators
generates the following output.

              TimeSpan           Duration( )             Negate( )
              --------           -----------             ---------
      00:00:00.0000001      00:00:00.0000001     -00:00:00.0000001
     -00:00:00.1234567      00:00:00.1234567      00:00:00.1234567
      00:09:39.9700000      00:09:39.9700000     -00:09:39.9700000
     -09:40:29.9600000      09:40:29.9600000      09:40:29.9600000
   -1.10:20:40.1600000    1.10:20:40.1600000    1.10:20:40.1600000
   10.20:30:40.0500000   10.20:30:40.0500000  -10.20:30:40.0500000
*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> System.TimeZone has been deprecated. Investigate the use of System.TimeZoneInfo instead.

Represents a time zone.

C#

```
[System.Obsolete("System.TimeZone has been deprecated. Investigate the use of System.TimeZoneInfo instead.")]
public abstract class TimeZone
```

Inheritance Object → TimeZone

Attributes ObsoleteAttribute

Examples

The following example references and displays selected `TimeZone` class elements.

C#

```
// Example of selected TimeZone class elements.
using System;
using System.Globalization;

class TimeZoneDemo
{
    static void Main( )
    {
        const string dataFmt = "{0,-30}{1}";
        const string timeFmt = "{0,-30}{1:yyyy-MM-dd HH:mm}";
```

```
        Console.WriteLine(
            "This example of selected TimeZone class " +
            "elements generates the following \n" +
            "output, which varies depending on the " +
            "time zone in which it is run.\n" );

        // Get the local time zone and the current local time and year.
        TimeZone localZone = TimeZone.CurrentTimeZone;
        DateTime currentDate = DateTime.Now;
        int      currentYear = currentDate.Year;

        // Display the names for standard time and daylight saving
        // time for the local time zone.
        Console.WriteLine( dataFmt, "Standard time name:",
            localZone.StandardName );
        Console.WriteLine( dataFmt, "Daylight saving time name:",
            localZone.DaylightName );

        // Display the current date and time and show if they occur
        // in daylight saving time.
        Console.WriteLine( "\n" + timeFmt, "Current date and time:",
            currentDate );
        Console.WriteLine( dataFmt, "Daylight saving time?",
            localZone.IsDaylightSavingTime( currentDate ) );

        // Get the current Coordinated Universal Time (UTC) and UTC
        // offset.
        DateTime currentUTC =
            localZone.ToUniversalTime( currentDate );
        TimeSpan currentOffset =
            localZone.GetUtcOffset( currentDate );

        Console.WriteLine( timeFmt, "Coordinated Universal Time:",
            currentUTC );
        Console.WriteLine( dataFmt, "UTC offset:", currentOffset );

        // Get the DaylightTime object for the current year.
        DaylightTime daylight =
            localZone.GetDaylightChanges( currentYear );

        // Display the daylight saving time range for the current year.
        Console.WriteLine(
            "\nDaylight saving time for year {0}:", currentYear );
        Console.WriteLine( "{0:yyyy-MM-dd HH:mm} to " +
            "{1:yyyy-MM-dd HH:mm}, delta: {2}",
            daylight.Start, daylight.End, daylight.Delta );
    }
}

/*
This example of selected TimeZone class elements generates the following
output, which varies depending on the time zone in which it is run.

Standard time name:           Pacific Standard Time
```

```
Daylight saving time name:    Pacific Daylight Time

Current date and time:        2006-01-06 16:47
Daylight saving time?         False
Coordinated Universal Time:   2006-01-07 00:47
UTC offset:                   -08:00:00

Daylight saving time for year 2006:
2006-04-02 02:00 to 2006-10-29 02:00, delta: 01:00:00
*/
```

Remarks

A time zone is a geographical region in which the same standard time is used.

> ⓘ **Important**
>
> Whenever possible, use the **TimeZoneInfo** class instead of the **TimeZone** class.

You can use the TimeZone class to retrieve information about the current time zone, and to convert times from local time to Coordinated Universal Time (UTC) or vice versa. However, you cannot use the TimeZone class to represent time zones other than the local zone or to handle date and time conversions from one time zone to another. For this purpose, use the TimeZoneInfo class. You can use this class to retrieve information on any time zone defined on the local system, to create custom time zones, and to convert times from one time zone to another.

The TimeZone class supports only a single daylight saving time adjustment rule for the local time zone. As a result, the TimeZone class can accurately report daylight saving time information or convert between UTC and local time only for the period in which the latest adjustment rule is in effect. In contrast, the TimeZoneInfo class supports multiple adjustment rules, which makes it possible to work with historic time zone data.

Notes to Implementers

In addition to providing implementations for its `abstract` members (those marked `MustOverride` in Visual Basic), it is important that classes derived from TimeZone override the default behavior of the ToLocalTime(DateTime) method. This is because the default behavior of ToLocalTime(DateTime) in the .NET Framework version 2.0 does not depend on a call to GetUtcOffset(DateTime), as it did in the .NET Framework versions 1.0 and 1.1. For details, see the ToLocalTime(DateTime) method.

Constructors

Expand table

TimeZone()	Initializes a new instance of the TimeZone class.

Properties

Expand table

CurrentTimeZone	Gets the time zone of the current computer.
DaylightName	Gets the daylight saving time zone name.
StandardName	Gets the standard time zone name.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetDaylightChanges(Int32)	Returns the daylight saving time period for a particular year.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetUtcOffset(DateTime)	Returns the Coordinated Universal Time (UTC) offset for the specified local time.
IsDaylightSavingTime(DateTime)	Returns a value indicating whether the specified date and time is within a daylight saving time period.
IsDaylightSavingTime(DateTime, DaylightTime)	Returns a value indicating whether the specified date and time is within the specified daylight saving time period.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToLocalTime(DateTime)	Returns the local time that corresponds to a specified date and

	time value.
ToString()	Returns a string that represents the current object. (Inherited from Object)
ToUniversalTime(DateTime)	Returns the Coordinated Universal Time (UTC) that corresponds to a specified time.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	*(2.0, 2.1)*

See also

- DateTime
- DateTimeOffset
- TimeZoneInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TimeZone class.

C#

```
protected TimeZone ();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone.CurrentTimeZone Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time zone of the current computer.

C#

```
public static TimeZone CurrentTimeZone { get; }
```

Property Value

TimeZone

A TimeZone object that represents the current local time zone.

Examples

The following example references the `CurrentTimeZone` property and displays the names for standard time and daylight saving time for the local time zone.

C#

```
// Example of selected TimeZone class elements.
using System;
using System.Globalization;

class TimeZoneDemo
{
    static void Main( )
    {
        const string dataFmt = "{0,-30}{1}";
        const string timeFmt = "{0,-30}{1:yyyy-MM-dd HH:mm}";

        Console.WriteLine(
            "This example of selected TimeZone class " +
            "elements generates the following \n" +
            "output, which varies depending on the " +
            "time zone in which it is run.\n" );
```

```
        // Get the local time zone and the current local time and year.
        TimeZone localZone = TimeZone.CurrentTimeZone;
        DateTime currentDate = DateTime.Now;
        int      currentYear = currentDate.Year;

        // Display the names for standard time and daylight saving
        // time for the local time zone.
        Console.WriteLine( dataFmt, "Standard time name:",
            localZone.StandardName );
        Console.WriteLine( dataFmt, "Daylight saving time name:",
            localZone.DaylightName );

        // Display the current date and time and show if they occur
        // in daylight saving time.
        Console.WriteLine( "\n" + timeFmt, "Current date and time:",
            currentDate );
        Console.WriteLine( dataFmt, "Daylight saving time?",
            localZone.IsDaylightSavingTime( currentDate ) );

        // Get the current Coordinated Universal Time (UTC) and UTC
        // offset.
        DateTime currentUTC =
            localZone.ToUniversalTime( currentDate );
        TimeSpan currentOffset =
            localZone.GetUtcOffset( currentDate );

        Console.WriteLine( timeFmt, "Coordinated Universal Time:",
            currentUTC );
        Console.WriteLine( dataFmt, "UTC offset:", currentOffset );

        // Get the DaylightTime object for the current year.
        DaylightTime daylight =
            localZone.GetDaylightChanges( currentYear );

        // Display the daylight saving time range for the current year.
        Console.WriteLine(
            "\nDaylight saving time for year {0}:", currentYear );
        Console.WriteLine( "{0:yyyy-MM-dd HH:mm} to " +
            "{1:yyyy-MM-dd HH:mm}, delta: {2}",
            daylight.Start, daylight.End, daylight.Delta );
    }
}

/*
This example of selected TimeZone class elements generates the following
output, which varies depending on the time zone in which it is run.

Standard time name:           Pacific Standard Time
Daylight saving time name:    Pacific Daylight Time

Current date and time:        2006-01-06 16:47
Daylight saving time?         False
Coordinated Universal Time:   2006-01-07 00:47
UTC offset:                   -08:00:00
```

```
Daylight saving time for year 2006:
2006-04-02 02:00 to 2006-10-29 02:00, delta: 01:00:00
*/
```

Remarks

The CurrentTimeZone property corresponds to the TimeZoneInfo.Local property. Whenever possible, use the TimeZoneInfo.Local property.

Notes to Callers

Local time zone data is cached after CurrentTimeZone is first used to retrieve time zone information. If the system's local time zone subsequently changes, the CurrentTimeZone property does not reflect this change. If you need to handle time zone changes while your application is running, use the TimeZoneInfo class and call its ClearCachedData() method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Local

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeZone.DaylightName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the daylight saving time zone name.

C#

```
public abstract string DaylightName { get; }
```

Property Value

String

The daylight saving time zone name.

Examples

The following example references and displays the `DaylightName` property of the current time zone.

C#

```
// Example of selected TimeZone class elements.
using System;
using System.Globalization;

class TimeZoneDemo
{
    static void Main( )
    {
        const string dataFmt = "{0,-30}{1}";
        const string timeFmt = "{0,-30}{1:yyyy-MM-dd HH:mm}";

        Console.WriteLine(
            "This example of selected TimeZone class " +
            "elements generates the following \n" +
            "output, which varies depending on the " +
            "time zone in which it is run.\n" );
```

```
        // Get the local time zone and the current local time and year.
        TimeZone localZone = TimeZone.CurrentTimeZone;
        DateTime currentDate = DateTime.Now;
        int      currentYear = currentDate.Year;

        // Display the names for standard time and daylight saving
        // time for the local time zone.
        Console.WriteLine( dataFmt, "Standard time name:",
            localZone.StandardName );
        Console.WriteLine( dataFmt, "Daylight saving time name:",
            localZone.DaylightName );

        // Display the current date and time and show if they occur
        // in daylight saving time.
        Console.WriteLine( "\n" + timeFmt, "Current date and time:",
            currentDate );
        Console.WriteLine( dataFmt, "Daylight saving time?",
            localZone.IsDaylightSavingTime( currentDate ) );

        // Get the current Coordinated Universal Time (UTC) and UTC
        // offset.
        DateTime currentUTC =
            localZone.ToUniversalTime( currentDate );
        TimeSpan currentOffset =
            localZone.GetUtcOffset( currentDate );

        Console.WriteLine( timeFmt, "Coordinated Universal Time:",
            currentUTC );
        Console.WriteLine( dataFmt, "UTC offset:", currentOffset );

        // Get the DaylightTime object for the current year.
        DaylightTime daylight =
            localZone.GetDaylightChanges( currentYear );

        // Display the daylight saving time range for the current year.
        Console.WriteLine(
            "\nDaylight saving time for year {0}:", currentYear );
        Console.WriteLine( "{0:yyyy-MM-dd HH:mm} to " +
            "{1:yyyy-MM-dd HH:mm}, delta: {2}",
            daylight.Start, daylight.End, daylight.Delta );
    }
}

/*
This example of selected TimeZone class elements generates the following
output, which varies depending on the time zone in which it is run.

Standard time name:           Pacific Standard Time
Daylight saving time name:    Pacific Daylight Time

Current date and time:        2006-01-06 16:47
Daylight saving time?         False
Coordinated Universal Time:   2006-01-07 00:47
UTC offset:                   -08:00:00
```

```
Daylight saving time for year 2006:
2006-04-02 02:00 to 2006-10-29 02:00, delta: 01:00:00
*/
```

Remarks

If daylight saving time is not used in the time zone, an empty string ("") is returned.

The DaylightName property corresponds to the TimeZoneInfo.DaylightName property. Whenever possible, use the TimeZoneInfo.DaylightName property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- DaylightName

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone.StandardName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the standard time zone name.

C#

```
public abstract string StandardName { get; }
```

Property Value

String

The standard time zone name.

Exceptions

ArgumentNullException

An attempt was made to set this property to `null`.

Examples

The following example references and displays the `StandardName` property of the current time zone.

C#

```
// Example of selected TimeZone class elements.
using System;
using System.Globalization;

class TimeZoneDemo
{
    static void Main( )
    {
        const string dataFmt = "{0,-30}{1}";
        const string timeFmt = "{0,-30}{1:yyyy-MM-dd HH:mm}";
```

```
        Console.WriteLine(
            "This example of selected TimeZone class " +
            "elements generates the following \n" +
            "output, which varies depending on the " +
            "time zone in which it is run.\n" );

        // Get the local time zone and the current local time and year.
        TimeZone localZone = TimeZone.CurrentTimeZone;
        DateTime currentDate = DateTime.Now;
        int      currentYear = currentDate.Year;

        // Display the names for standard time and daylight saving
        // time for the local time zone.
        Console.WriteLine( dataFmt, "Standard time name:",
            localZone.StandardName );
        Console.WriteLine( dataFmt, "Daylight saving time name:",
            localZone.DaylightName );

        // Display the current date and time and show if they occur
        // in daylight saving time.
        Console.WriteLine( "\n" + timeFmt, "Current date and time:",
            currentDate );
        Console.WriteLine( dataFmt, "Daylight saving time?",
            localZone.IsDaylightSavingTime( currentDate ) );

        // Get the current Coordinated Universal Time (UTC) and UTC
        // offset.
        DateTime currentUTC =
            localZone.ToUniversalTime( currentDate );
        TimeSpan currentOffset =
            localZone.GetUtcOffset( currentDate );

        Console.WriteLine( timeFmt, "Coordinated Universal Time:",
            currentUTC );
        Console.WriteLine( dataFmt, "UTC offset:", currentOffset );

        // Get the DaylightTime object for the current year.
        DaylightTime daylight =
            localZone.GetDaylightChanges( currentYear );

        // Display the daylight saving time range for the current year.
        Console.WriteLine(
            "\nDaylight saving time for year {0}:", currentYear );
        Console.WriteLine( "{0:yyyy-MM-dd HH:mm} to " +
            "{1:yyyy-MM-dd HH:mm}, delta: {2}",
            daylight.Start, daylight.End, daylight.Delta );
    }
}

/*
This example of selected TimeZone class elements generates the following
output, which varies depending on the time zone in which it is run.

Standard time name:           Pacific Standard Time
Daylight saving time name:    Pacific Daylight Time
```

```
Current date and time:        2006-01-06 16:47
Daylight saving time?         False
Coordinated Universal Time:   2006-01-07 00:47
UTC offset:                   -08:00:00

Daylight saving time for year 2006:
2006-04-02 02:00 to 2006-10-29 02:00, delta: 01:00:00
*/
```

Remarks

The StandardName property corresponds to the TimeZoneInfo.StandardName property. Whenever possible, use the TimeZoneInfo.StandardName property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- StandardName

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone.GetDaylightChanges(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the daylight saving time period for a particular year.

```
C#

public abstract System.Globalization.DaylightTime GetDaylightChanges (int
year);
```

Parameters

year Int32

The year that the daylight saving time period applies to.

Returns

DaylightTime

A DaylightTime object that contains the start and end date for daylight saving time in `year`.

Exceptions

ArgumentOutOfRangeException

`year` is less than 1 or greater than 9999.

Remarks

Only one daylight saving time period per year is supported.

If daylight saving time is not used in the current time zone, the return value is a DaylightTime object, where the value of Start and End is DateTime.MinValue, and the value of DaylightTime.Delta is a TimeSpan initialized to 0 ticks.

Notes to Callers

Because the TimeZone class supports only one daylight saving time adjustment rule, the GetDaylightChanges(Int32) method applies the current adjustment rule to any year, regardless of whether the adjustment rule actually applies to that year. Assuming that the operating system itself has accurate data for a particular year, a more accurate result is available by working with the array of TimeZoneInfo.AdjustmentRule objects returned by the GetAdjustmentRules() method. The Start, End, and Delta properties correspond to the DateStart, DateEnd, and DaylightDelta properties.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetAdjustmentRules()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone.GetUtcOffset(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the Coordinated Universal Time (UTC) offset for the specified local time.

```
C#
public abstract TimeSpan GetUtcOffset (DateTime time);
```

Parameters

`time` DateTime

A date and time value.

Returns

TimeSpan

The Coordinated Universal Time (UTC) offset from `time`.

Remarks

This method returns the offset, or difference, between the `time` parameter and Coordinated Universal Time (UTC). That is:

`time` = UTC + offset

The method interprets the time zone of `time` based on its Kind property. If the value of the Kind property is DateTimeKind.Local or DateTimeKind.Unspecified, the method returns the offset of the local time zone. If the value of the Kind property is DateTimeKind.Utc, the method returns an offset equal to TimeSpan.Zero.

If the local time zone observes daylight saving time, GetUtcOffset applies the current adjustment rule to `time` when determining the offset of the local time zone. That is, the

offset returned by GetUtcOffset reflects whether `time` falls in the time zone's standard time or its daylight saving time.

> **Note**
>
> The **GetUtcOffset** method recognizes only the current daylight saving time adjustment rule for the local time zone. As a result, it is guaranteed to accurately return the UTC offset of a local time only during the period in which the latest adjustment rule is in effect. It may return inaccurate results if `time` is a historic date and time value that was subject to a previous adjustment rule.

The GetUtcOffset method corresponds to the TimeZoneInfo.GetUtcOffset method. Whenever possible, use the TimeZoneInfo.GetUtcOffset method.

Because the date and time value represented by `time` and this value's offset from UTC are not tightly coupled, a local or unspecified date and time value can return a different offset value when run on different computers or when run on the same computer under different time zones. If this behavior is undesirable, use a DateTimeOffset value instead. The DateTimeOffset data type tightly couples a date and time value with its offset from UTC.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetUtcOffset(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeZone.IsDaylightSavingTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether a specified date and time is within a daylight saving time period.

Overloads

Expand table

IsDaylightSavingTime(DateTime)	Returns a value indicating whether the specified date and time is within a daylight saving time period.
IsDaylightSavingTime(DateTime, DaylightTime)	Returns a value indicating whether the specified date and time is within the specified daylight saving time period.

IsDaylightSavingTime(DateTime)

Returns a value indicating whether the specified date and time is within a daylight saving time period.

C#

```
public virtual bool IsDaylightSavingTime (DateTime time);
```

Parameters

`time` DateTime

A date and time.

Returns

Boolean

`true` if `time` is in a daylight saving time period; otherwise, `false`.

Remarks

The year to which the daylight saving time period applies is derived from the `time` parameter.

Because the TimeZone class supports a single daylight saving time adjustment rule, the IsDaylightSavingTime(DateTime) method applies the current adjustment rule to any date, regardless of whether the adjustment rule was in effect on that date. Assuming that the operating system itself has accurate historic daylight saving time data, a more accurate result is available by using the TimeZoneInfo.IsDaylightSavingTime method. Whenever possible, use the TimeZoneInfo.IsDaylightSavingTime method.

See also

- IsDaylightSavingTime(DateTime)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

IsDaylightSavingTime(DateTime, DaylightTime)

Returns a value indicating whether the specified date and time is within the specified daylight saving time period.

C#

```
public static bool IsDaylightSavingTime (DateTime time,
```

```
System.Globalization.DaylightTime daylightTimes);
```

Parameters

`time` DateTime

A date and time.

`daylightTimes` DaylightTime

A daylight saving time period.

Returns

Boolean

`true` if `time` is in `daylightTimes`; otherwise, `false`.

Exceptions

ArgumentNullException

`daylightTimes` is `null`.

Remarks

The TimeZoneInfo.IsDaylightSavingTime method provides the same functionality as this overload of the IsDaylightSavingTime method. Whenever possible, use the TimeZoneInfo.IsDaylightSavingTime method.

See also

- IsDaylightSavingTime(DateTime)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZone.ToLocalTime(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the local time that corresponds to a specified date and time value.

C#

```
public virtual DateTime ToLocalTime (DateTime time);
```

Parameters

`time` DateTime

A Coordinated Universal Time (UTC) time.

Returns

DateTime

A DateTime object whose value is the local time that corresponds to `time`.

Remarks

The following table shows the relationship between the `time` parameter and the DateTime value returned by this method.

Expand table

`time` parameter	Behavior	Return value
A Coordinated Universal Time (UTC) time (DateTimeKind.Utc).	Converts the time from UTC to the local time.	A DateTime object whose value is the local time that corresponds to `time`.

`time` parameter	Behavior	Return value
A local time (DateTimeKind.Local).	No conversion necessary.	The same DateTime value represented by the `time` parameter.
An unspecified time (DateTimeKind.Unspecified).	Assumes that the time is UTC and converts it from UTC to the local time.	A DateTime object whose value is the local time that corresponds to `time`.

If the local time zone observes daylight saving time, ToLocalTime applies the current adjustment rule to `time` when performing the conversion.

> **Note**
>
> The **ToLocalTime** method recognizes only the current daylight saving time adjustment rule for the local time zone. As a result, it is guaranteed to accurately return the local time corresponding to a particular UTC time only during the period in which the latest adjustment rule is in effect. It may return inaccurate results if `time` is a historic date and time value that was subject to a previous adjustment rule.

The ToLocalTime method corresponds to the TimeZoneInfo.ConvertTimeFromUtc method with its `destinationTimeZone` parameter set to TimeZoneInfo.Local. Whenever possible, use the TimeZoneInfo.ConvertTimeFromUtc method.

Notes to Inheritors

Although it is not required, in most cases derived classes running under the .NET Framework version 2.0 should override the default implementation of this method. In the .NET Framework versions 1.0 and 1.1, the `ToLocalTime` method called the GetUtcOffset(DateTime) method and adjusted for daylight saving time when returning the local time. However, starting with the .NET Framework 2.0, the behavior of the default implementation depends on the Kind property of the `time` parameter. If its value is Local, this method returns `time` unchanged. If its value is either Utc or Unspecified, this method assumes `time` is UTC and converts it to the local system time without calling the GetUtcOffset(DateTime) method.

The following code provides a simple override of the default implementation of the `ToLocalTime` method. In this code, the `internalTimeZone` variable represents a private instance of the `TimeZone` class:

C#

```
public override DateTime ToLocalTime(DateTime time)
{
   if (time.Kind == DateTimeKind.Local)
   {
      return time;
   }
   else if (time.Kind == DateTimeKind.Utc)
   {
      DateTime returnTime = new DateTime(time.Ticks, DateTimeKind.Local);
      returnTime += this.GetUtcOffset(returnTime);
      if (internalTimeZone.IsDaylightSavingTime(returnTime))
         returnTime -= new TimeSpan(1, 0, 0);
      return returnTime;
   }
   else
   {
      throw new ArgumentException("The source time zone cannot be
determined.");
   }
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ConvertTimeFromUtc(DateTime, TimeZoneInfo)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

TimeZone.ToUniversalTime(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the Coordinated Universal Time (UTC) that corresponds to a specified time.

C#

```
public virtual DateTime ToUniversalTime (DateTime time);
```

Parameters

`time` DateTime

A date and time.

Returns

DateTime

A DateTime object whose value is the Coordinated Universal Time (UTC) that corresponds to `time`.

Remarks

If the local time zone observes daylight saving time, ToUniversalTime applies the current adjustment rule to the `time` parameter when performing the conversion.

> **Note**
>
> The **ToUniversalTime** method recognizes only the current daylight saving time adjustment rule for the local time zone. As a result, it is guaranteed to accurately return the Coordinated Universal Time (UTC) corresponding to a particular local time only during the period in which the latest adjustment rule is in effect. It may

return inaccurate results if `time` is a historic date and time value that was subject to a previous adjustment rule.

If the `time` parameter is an ambiguous time, the method assumes that it is a standard time. (An ambiguous time is one that can map either to a standard time or to a daylight saving time in the local time zone.) If `time` is an invalid time, the method simply subtracts the local time from the local time zone's UTC offset to return UTC. (An invalid time is one that does not exist because of the application of daylight saving time adjustment rules.)

Because `time` is interpreted in relation to the current time zone on the current system, the date and time returned by this method can differ if an application is run on different computers or on the same computer with different time zones. For cases in which a date and time value must represent a single, unambiguous point in time, use a DateTimeOffset value to represent the local time.

The ToUniversalTime method corresponds to the TimeZoneInfo.ConvertTimeToUtc(DateTime) method overload with a DateTime parameter whose Kind property does not equal DateTimeKind.Utc. Whenever possible, use the TimeZoneInfo.ConvertTimeToUtc(DateTime) method overload.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ConvertTimeToUtc(DateTime)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeZoneInfo Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents any time zone in the world.

C#

```
public sealed class TimeZoneInfo : IEquatable<TimeZoneInfo>,
System.Runtime.Serialization.IDeserializationCallback,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → TimeZoneInfo

Implements IEquatable<TimeZoneInfo> , IDeserializationCallback , ISerializable

Remarks

A time zone is a geographical region in which the same time is used.

The TimeZoneInfo class offers significant enhancements over the TimeZone class, which provides only limited functionality.

The TimeZone class recognizes only the local time zone, and can convert times between Coordinated Universal Time (UTC) and local time. A TimeZoneInfo object can represent any time zone, and methods of the TimeZoneInfo class can be used to convert the time in one time zone to the corresponding time in any other time zone. The members of the TimeZoneInfo class support the following operations:

- Retrieving a time zone that is already defined by the operating system.
- Enumerating the time zones that are available on a system.
- Converting times between different time zones.
- Creating a new time zone that is not already defined by the operating system.
- Serializing a time zone for later retrieval.

> **Note**
>
> An instance of the **TimeZoneInfo** class is immutable. Once an object has been instantiated, its values cannot be modified.

You cannot instantiate a TimeZoneInfo object using the `new` keyword. Instead, you must call one of the static members of the TimeZoneInfo class shown in the following table.

Expand table

Static member name	Description
CreateCustomTimeZone method	Creates a custom time zone from application-supplied data.
FindSystemTimeZoneById method	Instantiates a time zone based on its identifier.
FromSerializedString method	Deserializes a string value to re-create a previously serialized TimeZoneInfo object.
GetSystemTimeZones method	Returns an enumerable ReadOnlyCollection<T> of TimeZoneInfo objects that represents all time zones that are available on the local system.
Local property	Instantiates a TimeZoneInfo object that represents the local time zone.
Utc property	Instantiates a TimeZoneInfo object that represents the UTC zone.

You can use the CreateCustomTimeZone method to create a time zone that is not defined in the local system registry on Windows systems or by the ICU Library's Time Zone Data on Linux or macOS. You can then use the ToSerializedString property to save the time zone object's information as a string, which can be stored in some form that is accessible to the application. You can use the FromSerializedString method to convert a serialized string back to a TimeZoneInfo object.

Properties

Expand table

BaseUtcOffset	Gets the time difference between the current time zone's standard time and Coordinated Universal Time (UTC).

DaylightName	Gets the display name for the current time zone's daylight saving time.
DisplayName	Gets the general display name that represents the time zone.
HasIanaId	Returns `true` if this TimeZoneInfo object has an IANA ID.
Id	Gets the time zone identifier.
Local	Gets a TimeZoneInfo object that represents the local time zone.
StandardName	Gets the display name for the time zone's standard time.
SupportsDaylightSavingTime	Gets a value indicating whether the time zone has any daylight saving time rules.
Utc	Gets a TimeZoneInfo object that represents the Coordinated Universal Time (UTC) zone.

Methods

Expand table

ClearCachedData()	Clears cached time zone data.
ConvertTime(DateTime, TimeZoneInfo)	Converts a time to the time in a particular time zone.
ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo)	Converts a time from one time zone to another.
ConvertTime(DateTimeOffset, TimeZoneInfo)	Converts a time to the time in a particular time zone.
ConvertTimeBySystemTimeZoneId(DateTime, String)	Converts a time to the time in another time zone based on the time zone's identifier.
ConvertTimeBySystemTimeZoneId(DateTime, String, String)	Converts a time from one time zone to another based on time zone identifiers.
ConvertTimeBySystemTimeZoneId(DateTimeOffset, String)	Converts a time to the time in another time zone based on the time zone's identifier.
ConvertTimeFromUtc(DateTime, TimeZoneInfo)	Converts a Coordinated Universal Time (UTC) to the time in a specified time zone.
ConvertTimeToUtc(DateTime)	Converts the specified date and time to Coordinated Universal Time (UTC).
ConvertTimeToUtc(DateTime, Time	Converts the time in a specified time zone to Coordinated

ZoneInfo)	Universal Time (UTC).
CreateCustomTimeZone(String, TimeSpan, String, String)	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, and a standard time display name.
CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[])	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, and daylight saving time rules.
CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[], Boolean)	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, daylight saving time rules, and a value that indicates whether the returned object reflects daylight saving time information.
Equals(Object)	Determines whether the current TimeZoneInfo object and another object are equal.
Equals(TimeZoneInfo)	Determines whether the current TimeZoneInfo object and another TimeZoneInfo object are equal.
FindSystemTimeZoneById(String)	Instantiates a new TimeZoneInfo object based on its identifier.
FromSerializedString(String)	Deserializes a string to re-create an original serialized TimeZoneInfo object.
GetAdjustmentRules()	Retrieves an array of TimeZoneInfo.AdjustmentRule objects that apply to the current TimeZoneInfo object.
GetAmbiguousTimeOffsets(DateTime)	Returns information about the possible dates and times that an ambiguous date and time can be mapped to.
GetAmbiguousTimeOffsets(DateTimeOffset)	Returns information about the possible dates and times that an ambiguous date and time can be mapped to.
GetHashCode()	Serves as a hash function for hashing algorithms and data structures such as hash tables.
GetSystemTimeZones()	Returns a sorted collection of all the time zones about which information is available on the local system.
GetSystemTimeZones(Boolean)	Returns a ReadOnlyCollection<T> containing all valid TimeZone's from the local machine. This method does *not* throw TimeZoneNotFoundException or InvalidTimeZoneException.
GetType()	Gets the Type of the current instance. (Inherited from Object)

GetUtcOffset(DateTime)	Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.
GetUtcOffset(DateTimeOffset)	Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.
HasSameRules(TimeZoneInfo)	Indicates whether the current object and another TimeZoneInfo object have the same adjustment rules.
IsAmbiguousTime(DateTime)	Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.
IsAmbiguousTime(DateTime Offset)	Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.
IsDaylightSavingTime(DateTime)	Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.
IsDaylightSavingTime(DateTime Offset)	Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.
IsInvalidTime(DateTime)	Indicates whether a particular date and time is invalid.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToSerializedString()	Converts the current TimeZoneInfo object to a serialized string.
ToString()	Returns the current TimeZoneInfo object's display name.
TryConvertIanaIdToWindows Id(String, String)	Tries to convert an IANA time zone ID to a Windows ID.
TryConvertWindowsIdToIana Id(String, String)	Tries to convert a Windows time zone ID to an IANA ID.
TryConvertWindowsIdToIana Id(String, String, String)	Tries to convert a Windows time zone ID to an IANA ID.
TryFindSystemTimeZone ById(String, TimeZoneInfo)	Retrieves a TimeZoneInfo object by time zone name.

Explicit Interface Implementations

Expand table

IDeserializationCallback.OnDeserialization(Object)	Runs when the deserialization of an object has been completed.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data needed to serialize the current TimeZoneInfo object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.BaseUtcOffset Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time difference between the current time zone's standard time and Coordinated Universal Time (UTC).

C#

```
public TimeSpan BaseUtcOffset { get; }
```

Property Value

TimeSpan

An object that indicates the time difference between the current time zone's standard time and Coordinated Universal Time (UTC).

Examples

The following example uses the BaseUtcOffset property to display the difference between the local time and Coordinated Universal Time (UTC).

C#

```
TimeZoneInfo localZone = TimeZoneInfo.Local;
Console.WriteLine("The {0} time zone is {1}:{2} {3} than Coordinated
Universal Time.",
                  localZone.DisplayName,
                  Math.Abs(localZone.BaseUtcOffset.Hours),
                  Math.Abs(localZone.BaseUtcOffset.Minutes),
                  (localZone.BaseUtcOffset >= TimeSpan.Zero) ? "later" :
"earlier");
```

Remarks

The time span returned by the BaseUtcOffset property can range from 14 hours (for a time zone that is 14 hours ahead of Coordinated Universal Time (UTC)) to -14 hours (for a time zone that is 14 hours behind UTC). Time zones that are ahead of UTC have a positive offset; time zones that are behind UTC have a negative offset.

The BaseUtcOffset value is represented as a whole number of minutes. It cannot include a fractional number of minutes.

> ⓘ **Note**
>
> Because **BaseUtcOffset** is a property of the **TimeZoneInfo** object rather than the **TimeZoneInfo.AdjustmentRule** object, the **TimeZoneInfo** class applies a single offset from UTC to all of a time zone's adjustments. To reflect a time zone that has modified its offset from UTC, you must create a new time zone using the **CreateCustomTimeZone** method.

The BaseUtcOffset property differs from the GetUtcOffset method in the following ways:

- The BaseUtcOffset property returns the difference between UTC and the time zone's standard time; the GetUtcOffset method returns the difference between UTC and the time zone's time at a particular point in time.
- The GetUtcOffset method reflects the application of any adjustment rules to the time zone; the BaseUtcOffset property does not.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GetUtcOffset

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.DaylightName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the display name for the current time zone's daylight saving time.

C#

```
public string DaylightName { get; }
```

Property Value

String

The display name for the time zone's daylight saving time.

Examples

The following example defines a method named `DisplayDateWithTimeZoneName` that uses the IsDaylightSavingTime(DateTime) method to determine whether to display a time zone's standard time name or daylight saving time name.

C#

```
private void DisplayDateWithTimeZoneName(DateTime date1, TimeZoneInfo
timeZone)
{
   Console.WriteLine("The time is {0:t} on {0:d} {1}",
                     date1,
                     timeZone.IsDaylightSavingTime(date1) ?
                         timeZone.DaylightName : timeZone.StandardName);
}
// The example displays output similar to the following:
//    The time is 1:00 AM on 4/2/2006 Pacific Standard Time
```

Remarks

The display name is localized based on the culture installed with the Windows operating system.

A DaylightName property whose value is not String.Empty or `null` does not necessarily indicate that the time zone supports daylight saving time. To determine whether the time zone supports daylight saving time, check the value of its SupportsDaylightSavingTime property.

In most cases, the `DaylightName` property of system-defined time zones is not String.Empty or `null`. However, the DaylightName property of custom time zones can be set to String.Empty. This occurs when custom time zones are created by the TimeZoneInfo.CreateCustomTimeZone(String, TimeSpan, String, String) or the TimeZoneInfo.CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[], Boolean) overload and the `disableDaylightSavingTime` parameter is `true`. Therefore, your code should never assume that the value of the DaylightName property is not `null` or empty.

The DaylightName property is equivalent to the DaylightName property of the TimeZone class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.DisplayName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the general display name that represents the time zone.

C#

```
public string DisplayName { get; }
```

Property Value

String

The time zone's general display name.

Examples

The following example retrieves a TimeZoneInfo object that represents the local time zone and outputs its display name, standard time name, and daylight saving time name. The output is displayed for a system in the U.S. Pacific Standard Time zone.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      TimeZoneInfo localZone = TimeZoneInfo.Local;
      Console.WriteLine("Local Time Zone ID: {0}", localZone.Id);
      Console.WriteLine("   Display Name is: {0}.", localZone.DisplayName);
      Console.WriteLine("   Standard name is: {0}.",
localZone.StandardName);
      Console.WriteLine("   Daylight saving name is: {0}.",
localZone.DaylightName);
   }
}
// The example displays output like the following:
```

```
//     Local Time Zone ID: Pacific Standard Time
//        Display Name is: (UTC-08:00) Pacific Time (US & Canada).
//        Standard name is: Pacific Standard Time.
//        Daylight saving name is: Pacific Daylight Time.
```

Remarks

The display name is localized based on the culture installed with the Windows operating system.

Time zone display names for Windows system time zones follow a fairly standard format. The first portion of the display name is the time zone's base offset from Coordinated Universal Time, which is indicated by the acronym GMT (for Greenwich Mean Time), enclosed in parentheses. For Coordinated Universal Time, the GMT acronym with no offset is enclosed in parentheses. This is followed by a string that identifies the time zone or one or more of the cities, regions, or countries in the time zone. For example:

```
(GMT) Greenwich Mean Time : Dublin, Edinburgh, Lisbon, London
(GMT+02:00) Athens, Beirut, Istanbul, Minsk
(GMT-02:00) Mid-Atlantic
(GMT-07:00) Mountain Time (US & Canada)
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeZoneInfo.HasIanaId Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns `true` if this TimeZoneInfo object has an IANA ID.

C#

```
public bool HasIanaId { get; }
```

Property Value

Boolean

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.Id Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time zone identifier.

```
C#
public string Id { get; }
```

Property Value

String

The time zone identifier.

Examples

The following example lists the identifier of each of the time zones defined on the local computer.

```
C#
ReadOnlyCollection<TimeZoneInfo> zones = TimeZoneInfo.GetSystemTimeZones();
Console.WriteLine("The local system has the following {0} time zones",
zones.Count);
foreach (TimeZoneInfo zone in zones)
   Console.WriteLine(zone.Id);
```

Remarks

The time zone identifier is a key string that uniquely identifies a particular time zone. On Windows systems, it corresponds to the subkeys of the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Time Zone branch of the registry on Windows systems and from the ICU Library on Linux and

macOS. It can be passed as a parameter to the FindSystemTimeZoneById method to instantiate a TimeZoneInfo object that represents a particular time zone.

> **Important**
>
> Although a key name in the Windows registry can be a maximum of 255 characters, we recommend that an identifier that you assign to a custom time zone be 32 characters or fewer. The longer the identifier, the greater the probability that an incorrect key name will be supplied when retrieving it.

The value of the Id property is usually, but not always, identical to that of the StandardName property. The identifier of the Coordinated Universal Time zone is UTC.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.Local Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a TimeZoneInfo object that represents the local time zone.

C#

```
public static TimeZoneInfo Local { get; }
```

Property Value

TimeZoneInfo

An object that represents the local time zone.

Examples

The following example retrieves a TimeZoneInfo object that represents the local time zone and outputs its display name, standard time name, and daylight saving time name.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      TimeZoneInfo localZone = TimeZoneInfo.Local;
      Console.WriteLine("Local Time Zone ID: {0}", localZone.Id);
      Console.WriteLine("   Display Name is: {0}.", localZone.DisplayName);
      Console.WriteLine("   Standard name is: {0}.",
localZone.StandardName);
      Console.WriteLine("   Daylight saving name is: {0}.",
localZone.DaylightName);
   }
}
// The example displays output like the following:
//     Local Time Zone ID: Pacific Standard Time
```

```
//        Display Name is: (UTC-08:00) Pacific Time (US & Canada).
//        Standard name is: Pacific Standard Time.
//        Daylight saving name is: Pacific Daylight Time.
```

Remarks

The local time zone is the time zone on the computer where the code is executing.

> ⓘ **Important**
>
> You should always access the local time zone through the **TimeZoneInfo.Local** property rather than assigning the local time zone to a **TimeZoneInfo** object variable. This prevents the **TimeZoneInfo** object variable from being invalidated by a call to the **ClearCachedData** method.

On Windows systems, the TimeZoneInfo object returned by the TimeZoneInfo.Local property reflects the setting of the **Automatically adjust clock for Daylight Saving Time** checkbox in the Control Panel **Date and Time** application. If the checkbox is unchecked, the cached copy of the local time zone contains no daylight saving time information. This means that:

- The local time zone's GetAdjustmentRules method returns an array whose length is zero.
- The local time zone's SupportsDaylightSavingTime property returns `false`.
- The local time zone has no ambiguous or invalid times (all calls to IsAmbiguousTime or IsInvalidTime return `false`).
- All calls to IsDaylightSavingTime with individual local times return false.

This is not true, however, if a reference to the local time zone is retrieved using the FindSystemTimeZoneById method.

The Local property corresponds to the CurrentTimeZone property of the TimeZone class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- How to: Access the Predefined UTC and Local Time Zone Objects

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.StandardName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the display name for the time zone's standard time.

C#

```
public string StandardName { get; }
```

Property Value

String

The display name of the time zone's standard time.

Examples

The following example defines a method named `DisplayDateWithTimeZoneName` that uses the IsDaylightSavingTime(DateTime) method to determine whether to display a time zone's standard time name or daylight saving time name.

C#

```
private void DisplayDateWithTimeZoneName(DateTime date1, TimeZoneInfo
timeZone)
{
   Console.WriteLine("The time is {0:t} on {0:d} {1}",
                     date1,
                     timeZone.IsDaylightSavingTime(date1) ?
                         timeZone.DaylightName : timeZone.StandardName);
}
// The example displays output similar to the following:
//    The time is 1:00 AM on 4/2/2006 Pacific Standard Time
```

Remarks

The display name is localized based on the culture installed with the Windows operating system.

The StandardName property is identical to the StandardName property of the TimeZone class.

If the operating system language is English, the value of the StandardName property is usually, but not always, identical to that of the Id property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.SupportsDaylightSaving Time Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the time zone has any daylight saving time rules.

C#

```
public bool SupportsDaylightSavingTime { get; }
```

Property Value

Boolean

`true` if the time zone supports daylight saving time; otherwise, `false`.

Examples

The following example retrieves a collection of all time zones that are available on a local system and displays the names of those that do not support daylight saving time.

C#

```
ReadOnlyCollection<TimeZoneInfo> zones = TimeZoneInfo.GetSystemTimeZones();
foreach(TimeZoneInfo zone in zones)
{
   if (! zone.SupportsDaylightSavingTime)
      Console.WriteLine(zone.DisplayName);
}
```

Remarks

The value of the SupportsDaylightSavingTime property for the local time zone returned by the TimeZoneInfo.Local property reflects the setting of the Control Panel **Date and Time** application's checkbox that defines whether the system automatically adjusts for

daylight saving time. If it is unchecked, or if no checkbox is displayed for a time zone, the value of this property is `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.Utc Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a TimeZoneInfo object that represents the Coordinated Universal Time (UTC) zone.

C#

```
public static TimeZoneInfo Utc { get; }
```

Property Value

TimeZoneInfo

An object that represents the Coordinated Universal Time (UTC) zone.

Examples

The following example retrieves a TimeZoneInfo object that represents Coordinated Universal Time (UTC) and outputs its display name, standard time name, and daylight saving time name.

C#

```
TimeZoneInfo universalZone = TimeZoneInfo.Utc;
Console.WriteLine("The universal time zone is {0}.",
universalZone.DisplayName);
Console.WriteLine("Its standard name is {0}.", universalZone.StandardName);
Console.WriteLine("Its daylight savings name is {0}.",
universalZone.DaylightName);
```

Remarks

This is a built-in object; information about this TimeZoneInfo object is not retrieved from the registry on Windows systems and from the ICU Library on Linux and macOS.

> **Important**
>
> You should always access the Coordinated Universal Time (UTC) zone through the **TimeZoneInfo.Utc** property rather than assigning the UTC time zone to a **TimeZoneInfo** object variable. This prevents the **TimeZoneInfo** object variable from being invalidated by a call to the **ClearCachedData** method.

Coordinated Universal Time was previously known as Greenwich Mean Time (GMT).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- How to: Access the Predefined UTC and Local Time Zone Objects

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ClearCachedData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clears cached time zone data.

C#

```
public static void ClearCachedData ();
```

Remarks

Cached time zone data includes data on the local time zone and the Coordinated Universal Time (UTC) zone.

You might call the `ClearCachedData` method to reduce the memory devoted to the application's cache of time zone information or to reflect the fact that the local system's time zone has changed.

Storing references to the local and UTC time zones is not recommended. After the call to the `ClearCachedData` method, these object variables will be undefined TimeZoneInfo objects that are no longer references to TimeZoneInfo.Local or TimeZoneInfo.Utc. For example, in the following code, the second call to the TimeZoneInfo.ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo) method throws an ArgumentException because the `local` variable is no longer considered equal to TimeZoneInfo.Local.

C#

```
TimeZoneInfo cst = TimeZoneInfo.FindSystemTimeZoneById("Central Standard
Time");
TimeZoneInfo local = TimeZoneInfo.Local;
Console.WriteLine(TimeZoneInfo.ConvertTime(DateTime.Now, local, cst));

TimeZoneInfo.ClearCachedData();
try
{
   Console.WriteLine(TimeZoneInfo.ConvertTime(DateTime.Now, local, cst));
```

```
}
catch (ArgumentException e)
{
   Console.WriteLine(e.GetType().Name + "\n   " + e.Message);
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ConvertTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a time to the time in a particular time zone.

Overloads

Expand table

ConvertTime(DateTime, TimeZoneInfo)	Converts a time to the time in a particular time zone.
ConvertTime(DateTimeOffset, TimeZoneInfo)	Converts a time to the time in a particular time zone.
ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo)	Converts a time from one time zone to another.

ConvertTime(DateTime, TimeZoneInfo)

Converts a time to the time in a particular time zone.

C#

```
public static DateTime ConvertTime (DateTime dateTime, TimeZoneInfo
destinationTimeZone);
```

Parameters

dateTime DateTime

The date and time to convert.

destinationTimeZone TimeZoneInfo

The time zone to convert `dateTime` to.

Returns

DateTime

The date and time in the destination time zone.

Exceptions

ArgumentException

The value of the `dateTime` parameter represents an invalid time.

ArgumentNullException

The value of the `destinationTimeZone` parameter is `null`.

Examples

The following example converts an array of date and time values to times in the Eastern Time zone of the U.S. and Canada. It shows that the source time zone depends on the DateTime.Kind property of the source DateTime value. It also illustrates that the ConvertTime method takes time zone adjustments into account, because a time zone adjustment occurs in both the source and destination time zones at 2:00 A.M. on November 7, 2010.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define times to be converted.
      DateTime[] times = { new DateTime(2010, 1, 1, 0, 1, 0),
                           new DateTime(2010, 1, 1, 0, 1, 0,
DateTimeKind.Utc),
                           new DateTime(2010, 1, 1, 0, 1, 0,
DateTimeKind.Local),
                           new DateTime(2010, 11, 6, 23, 30, 0),
                           new DateTime(2010, 11, 7, 2, 30, 0) };

      // Retrieve the time zone for Eastern Standard Time (U.S. and
Canada).
      TimeZoneInfo est;
      try {
         est = TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard
Time");
      }
      catch (TimeZoneNotFoundException) {
```

```
            Console.WriteLine("Unable to retrieve the Eastern Standard time
zone.");
            return;
        }
        catch (InvalidTimeZoneException) {
            Console.WriteLine("Unable to retrieve the Eastern Standard time
zone.");
            return;
        }

        // Display the current time zone name.
        Console.WriteLine("Local time zone: {0}\n",
TimeZoneInfo.Local.DisplayName);

        // Convert each time in the array.
        foreach (DateTime timeToConvert in times)
        {
            DateTime targetTime = TimeZoneInfo.ConvertTime(timeToConvert,
est);
            Console.WriteLine("Converted {0} {1} to {2}.", timeToConvert,
                              timeToConvert.Kind, targetTime);
        }
    }
}
// The example displays the following output:
//    Local time zone: (GMT-08:00) Pacific Time (US & Canada)
//
//    Converted 1/1/2010 12:01:00 AM Unspecified to 1/1/2010 3:01:00 AM.
//    Converted 1/1/2010 12:01:00 AM Utc to 12/31/2009 7:01:00 PM.
//    Converted 1/1/2010 12:01:00 AM Local to 1/1/2010 3:01:00 AM.
//    Converted 11/6/2010 11:30:00 PM Unspecified to 11/7/2010 1:30:00
AM.
//    Converted 11/7/2010 2:30:00 AM Unspecified to 11/7/2010 5:30:00 AM.
```

Remarks

When performing the conversion, the ConvertTime(DateTimeOffset, TimeZoneInfo) method applies any adjustment rules in effect in the `destinationTimeZone` time zone.

This overload of the ConvertTime(DateTime, TimeZoneInfo) method determines the source time zone from the value of the `dateTime` parameter's Kind property, as the following table shows.

Expand table

Kind property value	Source time zone	Method behavior
DateTimeKind.Local	Local	Converts the local time to the time in

Kind property value	Source time zone	Method behavior
		`destinationTimeZone`.
DateTimeKind.Utc	Utc	Converts Coordinated Universal Time (UTC) to the time in `destinationTimeZone`.
DateTimeKind.Unspecified	Assumed to be Local.	Converts the local time to the time in `destinationTimeZone`.

The Kind property of the returned DateTime value is set as shown in the following table.

Expand table

Condition	Returned Kind property value
The `destinationTimeZone` is TimeZoneInfo.Utc.	DateTimeKind.Utc
The `destinationTimeZone` is TimeZoneInfo.Local.	DateTimeKind.Local
All other date and time values and destination time zones.	DateTimeKind.Unspecified

If the value of the `dateTime` parameter is an ambiguous local time, it is interpreted as a standard time. If the `dateTime` parameter is an invalid local time, this method throws an ArgumentException.

If the conversion of `dateTime` results in a date and time value that is earlier than DateTime.MinValue or later than DateTime.MaxValue, this method returns DateTime.MinValue or DateTime.MaxValue, respectively.

You can also convert to or from UTC by calling the ConvertTimeFromUtc and ConvertTimeToUtc methods.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ConvertTime(DateTimeOffset, TimeZoneInfo)

Converts a time to the time in a particular time zone.

C#

```
public static DateTimeOffset ConvertTime (DateTimeOffset dateTimeOffset,
TimeZoneInfo destinationTimeZone);
```

Parameters

`dateTimeOffset` DateTimeOffset

The date and time to convert.

`destinationTimeZone` TimeZoneInfo

The time zone to convert `dateTime` to.

Returns

DateTimeOffset

The date and time in the destination time zone.

Exceptions

ArgumentNullException

The value of the `destinationTimeZone` parameter is `null`.

Examples

The following example converts an array of DateTimeOffset values to times in the Eastern Time zone of the U.S. and Canada. It illustrates that the ConvertTime method takes time zone adjustments into account, because a time zone adjustment occurs in both the source and destination time zones at 2:00 A.M. on November 7, 2010.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define times to be converted.
      DateTime time1 = new DateTime(2010, 1, 1, 12, 1, 0);
      DateTime time2 = new DateTime(2010, 11, 6, 23, 30, 0);
      DateTimeOffset[] times = { new DateTimeOffset(time1,
TimeZoneInfo.Local.GetUtcOffset(time1)),
                                 new DateTimeOffset(time1,
TimeSpan.Zero),
                                 new DateTimeOffset(time2,
TimeZoneInfo.Local.GetUtcOffset(time2)),
                                 new DateTimeOffset(time2.AddHours(3),
TimeZoneInfo.Local.GetUtcOffset(time2.AddHours(3))) };

      // Retrieve the time zone for Eastern Standard Time (U.S. and
Canada).
      TimeZoneInfo est;
      try {
         est = TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard
Time");
      }
      catch (TimeZoneNotFoundException) {
         Console.WriteLine("Unable to retrieve the Eastern Standard time
zone.");
         return;
      }
      catch (InvalidTimeZoneException) {
         Console.WriteLine("Unable to retrieve the Eastern Standard time
zone.");
         return;
      }

      // Display the current time zone name.
      Console.WriteLine("Local time zone: {0}\n",
TimeZoneInfo.Local.DisplayName);

      // Convert each time in the array.
      foreach (DateTimeOffset timeToConvert in times)
      {
         DateTimeOffset targetTime =
TimeZoneInfo.ConvertTime(timeToConvert, est);
         Console.WriteLine("Converted {0} to {1}.", timeToConvert,
```

```
targetTime);
        }
    }
}
// The example displays the following output:
//    Local time zone: (GMT-08:00) Pacific Time (US & Canada)
//
//    Converted 1/1/2010 12:01:00 AM -08:00 to 1/1/2010 3:01:00 AM
-05:00.
//    Converted 1/1/2010 12:01:00 AM +00:00 to 12/31/2009 7:01:00 PM
-05:00.
//    Converted 11/6/2010 11:30:00 PM -07:00 to 11/7/2010 1:30:00 AM
-05:00.
//    Converted 11/7/2010 2:30:00 AM -08:00 to 11/7/2010 5:30:00 AM
-05:00.
```

Remarks

When performing the conversion, the ConvertTime(DateTimeOffset, TimeZoneInfo) method applies any adjustment rules in effect in the `destinationTimeZone` time zone.

This overload differs from the other overloads of the ConvertTime method by accepting a DateTimeOffset value as its first parameter. This identifies the date and time as an offset from Coordinated Universal Time (UTC) rather than as the date and time in a particular time zone. As a result, the `dateTimeOffset` parameter cannot represent either an ambiguous time or an invalid time.

In converting the `dateTimeOffset` value to the time in the destination time zone, this method takes into account any adjustment rules in effect in the destination time zone.

If the conversion of `dateTimeOffset` results in a date and time value that is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue, this method returns DateTimeOffset.MinValue or DateTimeOffset.MaxValue, respectively.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo)

Converts a time from one time zone to another.

C#

```
public static DateTime ConvertTime (DateTime dateTime, TimeZoneInfo
sourceTimeZone, TimeZoneInfo destinationTimeZone);
```

Parameters

`dateTime` DateTime

The date and time to convert.

`sourceTimeZone` TimeZoneInfo

The time zone of `dateTime`.

`destinationTimeZone` TimeZoneInfo

The time zone to convert `dateTime` to.

Returns

DateTime

The date and time in the destination time zone that corresponds to the `dateTime` parameter in the source time zone.

Exceptions

ArgumentException

The Kind property of the `dateTime` parameter is Local, but the `sourceTimeZone` parameter does not equal Local.

-or-

The Kind property of the `dateTime` parameter is Utc, but the `sourceTimeZone` parameter does not equal Utc.

-or-

The `dateTime` parameter is an invalid time (that is, it represents a time that does not exist because of a time zone's adjustment rules).

ArgumentNullException

The `sourceTimeZone` parameter is `null`.

-or-

The `destinationTimeZone` parameter is `null`.

Examples

The following example illustrates the use of the ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo) method to convert from Hawaiian Standard Time to local time.

C#

```
DateTime hwTime = new DateTime(2007, 02, 01, 08, 00, 00);
try
{
   TimeZoneInfo hwZone = TimeZoneInfo.FindSystemTimeZoneById("Hawaiian
Standard Time");
   Console.WriteLine("{0} {1} is {2} local time.",
           hwTime,
           hwZone.IsDaylightSavingTime(hwTime) ? hwZone.DaylightName :
hwZone.StandardName,
           TimeZoneInfo.ConvertTime(hwTime, hwZone, TimeZoneInfo.Local));
}
catch (TimeZoneNotFoundException)
{
   Console.WriteLine("The registry does not define the Hawaiian Standard
Time zone.");
}
catch (InvalidTimeZoneException)
{
   Console.WriteLine("Registry data on the Hawaiian Standard Time zone
```

```
has been corrupted.");
}
```

Remarks

When performing the conversion, the ConvertTime method applies any adjustment rules in effect in the `destinationTimeZone` time zone.

The value of the Kind property of the `dateTime` parameter must correspond to the `sourceTimeZone` parameter, as the following table shows.

Expand table

DateTime.Kind value	sourceTimeZone value	Method behavior
DateTimeKind.Utc	Equals TimeZoneInfo.Utc.	Converts `dateTime` to the destination time zone's time.
DateTimeKind.Utc	Does not equal TimeZoneInfo.Utc.	Throws an ArgumentException.
DateTimeKind.Local	Equals TimeZoneInfo.Local.	Converts `dateTime` to the destination time zone's time.
DateTimeKind.Local	Does not equal TimeZoneInfo.Local.	Throws an ArgumentException.
DateTimeKind.Unspecified	Any.	Converts `dateTime` to the destination time zone's time.

You can also convert to or from Coordinated Universal Time (UTC) by calling the ConvertTimeFromUtc and ConvertTimeToUtc methods.

The Kind property of the returned DateTime value is set as shown in the following table.

Expand table

Condition	Returned Kind property value
The `destinationTimeZone` argument is TimeZoneInfo.Utc.	DateTimeKind.Utc
The `destinationTimeZone` argument is TimeZoneInfo.Local.	DateTimeKind.Local
All other date and time values, source time zones, and	DateTimeKind.Unspecified

Condition	Returned Kind property value
destination time zones.	

If the value of the `dateTime` parameter is an ambiguous time in the source time zone, it is interpreted as a standard time. If the `dateTime` parameter is an invalid time in the source time zone, this method throws an ArgumentException.

If the conversion of `dateTime` results in a date and time value that is earlier than DateTime.MinValue or later than DateTime.MaxValue, this method returns DateTime.MinValue or DateTime.MaxValue, respectively.

The ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo) method throws an ArgumentException exception if the DateTime.Kind property of the `dateTime` argument is DateTimeKind.Local but the `sourceTimeZone` argument is not TimeZoneInfo.Local. To determine whether the source time zone is the local time zone or the universal time zone, the method tests for reference equality instead of testing for value equality with the Equals(TimeZoneInfo) method. Note that TimeZoneInfo objects that represent the local time zone and that are retrieved by calling the FindSystemTimeZoneById method do not have referential equality with TimeZoneInfo.Local. Furthermore, TimeZoneInfo objects that represent the local or universal time zone and that are retrieved by iterating the collection returned by the GetSystemTimeZones method do not have referential equality with TimeZoneInfo.Local or TimeZoneInfo.Utc. As an alternative, you can call the ConvertTimeBySystemTimeZoneId(DateTime, String, String) method.

See also

- ConvertTimeBySystemTimeZoneId
- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ConvertTimeBySystemTimeZoneId Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a time to the time in another time zone based on a time zone identifier.

Overloads

Expand table

ConvertTimeBySystemTimeZoneId(DateTime, String)	Converts a time to the time in another time zone based on the time zone's identifier.
ConvertTimeBySystemTimeZoneId(DateTimeOffset, String)	Converts a time to the time in another time zone based on the time zone's identifier.
ConvertTimeBySystemTimeZoneId(DateTime, String, String)	Converts a time from one time zone to another based on time zone identifiers.

ConvertTimeBySystemTimeZoneId(DateTime, String)

Converts a time to the time in another time zone based on the time zone's identifier.

C#

```
public static DateTime ConvertTimeBySystemTimeZoneId (DateTime dateTime,
string destinationTimeZoneId);
```

Parameters

dateTime DateTime

The date and time to convert.

`destinationTimeZoneId` String

The identifier of the destination time zone.

Returns

DateTime

The date and time in the destination time zone.

Exceptions

ArgumentNullException

`destinationTimeZoneId` is `null`.

InvalidTimeZoneException

The time zone identifier was found, but the registry data is corrupted.

SecurityException

The process does not have the permissions required to read from the registry key that contains the time zone information.

TimeZoneNotFoundException

The `destinationTimeZoneId` identifier was not found on the local system.

Remarks

When performing the conversion, the ConvertTimeBySystemTimeZoneId method applies any adjustment rules in effect in the `destinationTimeZoneId` time zone.

This overload is largely identical to calling the ConvertTime(DateTime, TimeZoneInfo) method, except that it allows you to specify the destination time zone by its identifier rather than by an object reference. This method is most useful when you must convert a time without retrieving the time zone object that corresponds to it and you do not need to know whether the converted time is standard or daylight saving time.

The ConvertTimeBySystemTimeZoneId(DateTime, String) method determines the source time zone from the value of the `dateTime` parameter's Kind property, as the following table shows.

Expand table

Kind property value	Source time zone	Method behavior
DateTimeKind.Local	Local	Converts the local time to the time in `destinationTimeZone`.
DateTimeKind.Utc	Utc	Converts Coordinated Universal Time (UTC) to the time in `destinationTimeZone`.
DateTimeKind.Unspecified	Assumed to be Local.	Converts the local time to the time in `destinationTimeZone`.

The Kind property of the returned DateTime value is set as shown in the following table.

Expand table

Condition	Returned Kind property value
The `destinationTimeZone` is `TimeZoneInfo.Utc.Id`.	DateTimeKind.Utc
Any other `destinationTimeZone` value.	DateTimeKind.Unspecified

If the value of the `dateTime` parameter is an ambiguous local time, it is interpreted as a standard time. If the `dateTime` parameter is an invalid local time, this method throws an ArgumentException.

If the conversion of `dateTime` results in a date and time value that is earlier than DateTime.MinValue or later than DateTime.MaxValue, this method returns DateTime.MinValue or DateTime.MaxValue, respectively.

This method retrieves information on the time zone whose identifier is specified by the `destinationTimeZoneId` parameter from the registry on Windows systems and from the ICU Library on Linux and macOS. It cannot retrieve a time zone object that is created using the CreateCustomTimeZone method. The `destinationTimeZoneId` parameter must correspond exactly to the time zone's identifier in length, but not in case, for a successful match to occur; that is, the comparison of `destinationTimeZoneId` with time zone identifiers is case-insensitive.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ConvertTimeBySystemTimeZoneId(DateTimeOffset, String)

Converts a time to the time in another time zone based on the time zone's identifier.

C#

```
public static DateTimeOffset ConvertTimeBySystemTimeZoneId
(DateTimeOffset dateTimeOffset, string destinationTimeZoneId);
```

Parameters

`dateTimeOffset` DateTimeOffset

The date and time to convert.

`destinationTimeZoneId` String

The identifier of the destination time zone.

Returns

DateTimeOffset

The date and time in the destination time zone.

Exceptions

ArgumentNullException

`destinationTimeZoneId` is `null`.

InvalidTimeZoneException

The time zone identifier was found but the registry data is corrupted.

SecurityException
The process does not have the permissions required to read from the registry key that contains the time zone information.

TimeZoneNotFoundException
The `destinationTimeZoneId` identifier was not found on the local system.

Remarks

When performing the conversion, the ConvertTimeBySystemTimeZoneId method applies any adjustment rules in effect in the `destinationTimeZoneId` time zone.

This overload is identical to calling the ConvertTime(DateTimeOffset, TimeZoneInfo) method, except that it allows you to specify the destination time zone by its identifier rather than by an object reference. This method is most useful when you must convert a time without retrieving the time zone object that corresponds to it and you do not need to know whether the converted time is standard or daylight saving time.

Because the `dateTimeOffset` parameter represents a date and time together with that time's offset from Coordinated Universal Time (UTC), it cannot represent either an ambiguous time or an invalid time.

This method retrieves the time zone whose identifier is specified by the `destinationTimeZoneId` parameter from the registry on Windows systems and from the ICU Library on Linux and macOS. It cannot retrieve a time zone object that is created using the CreateCustomTimeZone method. The `destinationTimeZoneId` parameter must correspond exactly to the time zone's identifier in length, but not in case, for a successful match to occur; that is, the comparison of `destinationTimeZoneId` with time zone identifiers is case-insensitive.

In converting the `dateTimeOffset` value to the time in the destination time zone, the method takes into account any adjustment rules in effect in the destination time zone.

If the conversion of `dateTimeOffset` results in a date and time value that is earlier than DateTimeOffset.MinValue or later than DateTimeOffset.MaxValue, this method returns DateTimeOffset.MinValue or DateTimeOffset.MaxValue, respectively.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ConvertTimeBySystemTimeZoneId(DateTime, String, String)

Converts a time from one time zone to another based on time zone identifiers.

C#

```
public static DateTime ConvertTimeBySystemTimeZoneId (DateTime dateTime,
string sourceTimeZoneId, string destinationTimeZoneId);
```

Parameters

`dateTime` DateTime

The date and time to convert.

`sourceTimeZoneId` String

The identifier of the source time zone.

`destinationTimeZoneId` String

The identifier of the destination time zone.

Returns

DateTime

The date and time in the destination time zone that corresponds to the `dateTime` parameter in the source time zone.

Exceptions

ArgumentException

The Kind property of the `dateTime` parameter does not correspond to the source time zone.

-or-

`dateTime` is an invalid time in the source time zone.

ArgumentNullException

`sourceTimeZoneId` is `null`.

-or-

`destinationTimeZoneId` is `null`.

InvalidTimeZoneException

The time zone identifiers were found, but the registry data is corrupted.

SecurityException

The user does not have the permissions required to read from the registry keys that hold time zone data.

TimeZoneNotFoundException

The `sourceTimeZoneId` identifier was not found on the local system.

-or-

The `destinationTimeZoneId` identifier was not found on the local system.

Examples

The following example uses the TimeZoneInfo.ConvertTimeBySystemTimeZoneId(DateTime, String, String) method to display the time that corresponds to the local system time in eight cities of the world.

```
C#
```

```
DateTime currentTime = DateTime.Now;
Console.WriteLine("Current Times:");
Console.WriteLine();
Console.WriteLine("Los Angeles: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "Pacific Standard Time"));
Console.WriteLine("Chicago: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "Central Standard Time"));
Console.WriteLine("New York: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "Eastern Standard Time"));
Console.WriteLine("London: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "GMT Standard Time"));
Console.WriteLine("Moscow: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "Russian Standard Time"));
Console.WriteLine("New Delhi: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "India Standard Time"));
Console.WriteLine("Beijing: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "China Standard Time"));
Console.WriteLine("Tokyo: {0}",
                  TimeZoneInfo.ConvertTimeBySystemTimeZoneId(currentTime,
TimeZoneInfo.Local.Id, "Tokyo Standard Time"));
```

Remarks

When performing the conversion, the ConvertTimeBySystemTimeZoneId method applies any adjustment rules in effect in the `destinationTimeZoneId` time zone.

Although it is similar to the TimeZoneInfo.ConvertTime(DateTime, TimeZoneInfo, TimeZoneInfo) method, you can use TimeZoneInfo.ConvertTimeBySystemTimeZoneId(DateTime, String, String) to specify the source and destination time zones using their identifiers instead of their TimeZoneInfo objects. This method is most useful when you must convert a time without retrieving the time zone object that corresponds to it and you do not need to know whether the converted time is standard or daylight saving time.

This method retrieves the time zones whose identifiers are the `sourceTimeZoneId` and `destinationTimeZoneId` parameters from the registry on Windows systems and from the ICU Library on Linux and macOS. It cannot retrieve time zone objects that are created using the CreateCustomTimeZone method.

The value of the Kind property of the `dateTime` parameter must correspond to the `sourceTimeZoneId` parameter, as the following table shows.

Expand table

DateTime.Kind value	sourceTimeZone value	Method behavior
DateTimeKind.Utc	Equals `TimeZoneInfo.Utc.Id`.	Converts `dateTime` to the destination time zone's time.
DateTimeKind.Utc	Does not equal `TimeZoneInfo.Utc.Id`.	Throws an ArgumentException.
DateTimeKind.Local	Equals `TimeZoneInfo.Local.Id`.	Converts `dateTime` to the destination time zone's time.
DateTimeKind.Local	Does not equal `TimeZoneInfo.Local.Id`.	Throws an ArgumentException.
DateTimeKind.Unspecified	Any.	Converts `dateTime` to the destination time zone's time.

Because it relies on calls to the FindSystemTimeZoneById method, the ConvertTimeBySystemTimeZoneId method performs a case-insensitive search to locate the time zones that correspond to `sourceTimeZoneId` and `destinationTimeZoneId`.

If the value of the `dateTime` parameter is an ambiguous time in the source time zone, it is interpreted as a standard time. If the `dateTime` parameter is an invalid time in the source time zone, this method throws an ArgumentException.

The Kind property of the returned DateTime value is set to DateTimeKind.Unspecified unless the destination time zone is Coordinated Universal Time (UTC), in which case it is set to DateTimeKind.Utc.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.ConvertTimeFromUtc(DateTime, TimeZoneInfo) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a Coordinated Universal Time (UTC) to the time in a specified time zone.

C#

```
public static DateTime ConvertTimeFromUtc (DateTime dateTime, TimeZoneInfo
destinationTimeZone);
```

Parameters

`dateTime` DateTime

The Coordinated Universal Time (UTC).

`destinationTimeZone` TimeZoneInfo

The time zone to convert `dateTime` to.

Returns

DateTime

The date and time in the destination time zone. Its Kind property is Utc if `destinationTimeZone` is Utc; otherwise, its Kind property is Unspecified.

Exceptions

ArgumentException

The Kind property of `dateTime` is Local.

ArgumentNullException

`destinationTimeZone` is `null`.

Examples

The following example converts Coordinated Universal Time (UTC) to Central Time.

C#

```
DateTime timeUtc = DateTime.UtcNow;
try
{
   TimeZoneInfo cstZone = TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time");
   DateTime cstTime = TimeZoneInfo.ConvertTimeFromUtc(timeUtc, cstZone);
   Console.WriteLine("The date and time are {0} {1}.",
                     cstTime,
                     cstZone.IsDaylightSavingTime(cstTime) ?
                             cstZone.DaylightName : cstZone.StandardName);
}
catch (TimeZoneNotFoundException)
{
   Console.WriteLine("The registry does not define the Central Standard Time
zone.");
}
catch (InvalidTimeZoneException)
{
   Console.WriteLine("Registry data on the Central Standard Time zone has
been corrupted.");
}
```

Remarks

When performing the conversion, the ConvertTimeFromUtc method applies any adjustment rules in effect in the `destinationTimeZone` time zone.

The precise behavior of this method depends on the value of the Kind property of the `dateTime` parameter, as the following table shows.

Expand table

DateTime.Kind property	Conversion
DateTimeKind.Local	Throws an ArgumentException.
DateTimeKind.Unspecified or DateTimeKind.Utc	Converts from Coordinated Universal Time (UTC).

If the conversion of `dateTime` results in a date and time value that is earlier than DateTime.MinValue or later than DateTime.MaxValue, this method returns

DateTime.MinValue or DateTime.MaxValue, respectively.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Converting Times Between Time Zones

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ConvertTimeToUtc Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a date and time to Coordinated Universal Time (UTC).

Overloads

Expand table

ConvertTimeToUtc(DateTime)	Converts the specified date and time to Coordinated Universal Time (UTC).
ConvertTimeToUtc(DateTime, TimeZoneInfo)	Converts the time in a specified time zone to Coordinated Universal Time (UTC).

ConvertTimeToUtc(DateTime)

Converts the specified date and time to Coordinated Universal Time (UTC).

C#

```
public static DateTime ConvertTimeToUtc (DateTime dateTime);
```

Parameters

`dateTime` DateTime

The date and time to convert.

Returns

DateTime

The Coordinated Universal Time (UTC) that corresponds to the `dateTime` parameter. The DateTime value's Kind property is always set to Utc.

Exceptions

ArgumentException

`TimeZoneInfo.Local.IsInvalidDateTime( dateTime )` returns `true`.

Examples

The following example illustrates the conversion of time values whose Kind property is DateTimeKind.Utc, DateTimeKind.Local, and DateTimeKind.Unspecified, respectively. It also illustrates the conversion of ambiguous and invalid times.

C#

```
DateTime datNowLocal = DateTime.Now;
Console.WriteLine("Converting {0}, Kind {1}:", datNowLocal,
datNowLocal.Kind);
Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datNowLocal),
TimeZoneInfo.ConvertTimeToUtc(datNowLocal).Kind);
Console.WriteLine();

DateTime datNowUtc = DateTime.UtcNow;
Console.WriteLine("Converting {0}, Kind {1}", datNowUtc, datNowUtc.Kind);
Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datNowUtc),
TimeZoneInfo.ConvertTimeToUtc(datNowUtc).Kind);
Console.WriteLine();

DateTime datNow = new DateTime(2007, 10, 26, 13, 32, 00);
Console.WriteLine("Converting {0}, Kind {1}", datNow, datNow.Kind);
Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datNow),
TimeZoneInfo.ConvertTimeToUtc(datNow).Kind);
Console.WriteLine();

DateTime datAmbiguous = new DateTime(2007, 11, 4, 1, 30, 00);
Console.WriteLine("Converting {0}, Kind {1}, Ambiguous {2}",
datAmbiguous, datAmbiguous.Kind,
TimeZoneInfo.Local.IsAmbiguousTime(datAmbiguous));
Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datAmbiguous),
TimeZoneInfo.ConvertTimeToUtc(datAmbiguous).Kind);
Console.WriteLine();

DateTime datInvalid = new DateTime(2007, 3, 11, 02, 30, 00);
Console.WriteLine("Converting {0}, Kind {1}, Invalid {2}", datInvalid,
datInvalid.Kind, TimeZoneInfo.Local.IsInvalidTime(datInvalid));
```

```
try
{
   Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datInvalid),
TimeZoneInfo.ConvertTimeToUtc(datInvalid).Kind);
}
catch (ArgumentException e)
{
   Console.WriteLine("   {0}: Cannot convert {1} to UTC.",
e.GetType().Name, datInvalid);
}
Console.WriteLine();

DateTime datNearMax = new DateTime(9999, 12, 31, 22, 00, 00);
Console.WriteLine("Converting {0}, Kind {1}", datNearMax,
datNearMax.Kind);
Console.WriteLine("   ConvertTimeToUtc: {0}, Kind {1}",
TimeZoneInfo.ConvertTimeToUtc(datNearMax),
TimeZoneInfo.ConvertTimeToUtc(datNearMax).Kind);
Console.WriteLine();
//
// This example produces the following output if the local time zone
// is Pacific Standard Time:
//
//    Converting 8/31/2007 2:26:28 PM, Kind Local:
//       ConvertTimeToUtc: 8/31/2007 9:26:28 PM, Kind Utc
//
//    Converting 8/31/2007 9:26:28 PM, Kind Utc
//       ConvertTimeToUtc: 8/31/2007 9:26:28 PM, Kind Utc
//
//    Converting 10/26/2007 1:32:00 PM, Kind Unspecified
//       ConvertTimeToUtc: 10/26/2007 8:32:00 PM, Kind Utc
//
//    Converting 11/4/2007 1:30:00 AM, Kind Unspecified, Ambiguous True
//       ConvertTimeToUtc: 11/4/2007 9:30:00 AM, Kind Utc
//
//    Converting 3/11/2007 2:30:00 AM, Kind Unspecified, Invalid True
//       ArgumentException: Cannot convert 3/11/2007 2:30:00 AM to UTC.
//
//    Converting 12/31/9999 10:00:00 PM, Kind Unspecified
//       ConvertTimeToUtc: 12/31/9999 11:59:59 PM, Kind Utc
//
```

Remarks

The exact conversion performed depends on the value of the Kind property of the `dateTime` parameter, as the following table shows.

Expand table

DateTime.Kind property	Conversion
DateTimeKind.Local	Converts from local time to Coordinated Universal Time (UTC).
DateTimeKind.Unspecified	Assumes `dateTime` is local time and converts from local time to UTC.
DateTimeKind.Utc	Returns `dateTime` unchanged.

If `dateTime` corresponds to an ambiguous local time, this method assumes that it is standard local time. If `dateTime` corresponds to an invalid local time, the method throws an ArgumentException.

> ⓘ **Note**
>
> If the current computer's local time zone includes multiple adjustment rules, this overload of the **ConvertTimeToUtc** method can return results that differ from the **TimeZone.ToUniversalTime** and **DateTime.ToUniversalTime** methods. **TimeZone.ToUniversalTime** always applies the current adjustment rule to time zone conversion, whether or not `dateTime` lies within its date range. And when executing on .NET Framework 3.5, **DateTime.ToUniversalTime** also applies the current adjustment rule to time zone conversion, whether or not `dateTime` lies within its date range.

If the UTC equivalent of `dateTime` is earlier than DateTime.MinValue or later that DateTime.MaxValue, this method returns MinValue or MaxValue, respectively.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ConvertTimeToUtc(DateTime, TimeZoneInfo)

Converts the time in a specified time zone to Coordinated Universal Time (UTC).

```
C#

public static DateTime ConvertTimeToUtc (DateTime dateTime, TimeZoneInfo
sourceTimeZone);
```

Parameters

`dateTime` DateTime

The date and time to convert.

`sourceTimeZone` TimeZoneInfo

The time zone of `dateTime`.

Returns

DateTime

The Coordinated Universal Time (UTC) that corresponds to the `dateTime` parameter. The DateTime object's Kind property is always set to Utc.

Exceptions

ArgumentException

`dateTime.Kind` is Utc and `sourceTimeZone` does not equal Utc.

-or-

`dateTime.Kind` is Local and `sourceTimeZone` does not equal Local.

-or-

`sourceTimeZone.IsInvalidDateTime( dateTime )` returns `true`.

ArgumentNullException

`sourceTimeZone` is `null`.

Examples

The following example retrieves the current date from the local system and converts it to Coordinated Universal Time (UTC), then converts it to Tokyo Standard Time, and finally converts from Tokyo Standard Time back to UTC. Note that the two UTC times are identical.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get time in local time zone
      DateTime thisTime = DateTime.Now;
      Console.WriteLine("Time in {0} zone: {1}",
TimeZoneInfo.Local.IsDaylightSavingTime(thisTime) ?
                        TimeZoneInfo.Local.DaylightName :
TimeZoneInfo.Local.StandardName, thisTime);
      Console.WriteLine("   UTC Time: {0}",
TimeZoneInfo.ConvertTimeToUtc(thisTime, TimeZoneInfo.Local));
      // Get Tokyo Standard Time zone
      TimeZoneInfo tst = TimeZoneInfo.FindSystemTimeZoneById("Tokyo
Standard Time");
      DateTime tstTime = TimeZoneInfo.ConvertTime(thisTime,
TimeZoneInfo.Local, tst);
      Console.WriteLine("Time in {0} zone: {1}",
tst.IsDaylightSavingTime(tstTime) ?
                        tst.DaylightName : tst.StandardName, tstTime);
      Console.WriteLine("   UTC Time: {0}",
TimeZoneInfo.ConvertTimeToUtc(tstTime, tst));
   }
}
// The example displays output like the following when run on a system in
the
// U.S. Pacific Standard Time zone:
//       Time in Pacific Standard Time zone: 12/6/2013 10:57:51 AM
//          UTC Time: 12/6/2013 6:57:51 PM
//       Time in Tokyo Standard Time zone: 12/7/2013 3:57:51 AM
//          UTC Time: 12/6/2013 6:57:51 PM
```

Remarks

If the Kind property of the `dateTime` parameter equals DateTimeKind.Utc and the `sourceTimeZone` parameter equals TimeZoneInfo.Utc, this method returns `dateTime` without performing any conversion.

If `dateTime` corresponds to an ambiguous time, this method assumes that it is the standard time of the source time zone. If `dateTime` corresponds to an invalid time,

this method throws an ArgumentException.

If the Coordinated Universal Time (UTC) equivalent of `dateTime` is earlier than DateTime.MinValue or later that DateTime.MaxValue, this method returns MinValue or MaxValue, respectively.

See also

- Converting Times Between Time Zones

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.CreateCustomTimeZone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a time zone that is not found on the local computer.

Overloads

Expand table

CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[], Boolean)	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, daylight saving time rules, and a value that indicates whether the returned object reflects daylight saving time information.
CreateCustomTimeZone(String, TimeSpan, String, String)	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, and a standard time display name.
CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[])	Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, and daylight saving time rules.

CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[], Boolean)

Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, daylight saving time rules, and a value that indicates whether the returned object reflects daylight saving time information.

```
C#

public static TimeZoneInfo CreateCustomTimeZone (string id, TimeSpan
baseUtcOffset, string? displayName, string? standardDisplayName, string?
daylightDisplayName, TimeZoneInfo.AdjustmentRule[]? adjustmentRules, bool
disableDaylightSavingTime);
```

Parameters

`id` String

The time zone's identifier.

`baseUtcOffset` TimeSpan

A TimeSpan object that represents the time difference between this time zone and Coordinated Universal Time (UTC).

`displayName` String

The display name of the new time zone.

`standardDisplayName` String

The standard time name of the new time zone.

`daylightDisplayName` String

The daylight saving time name of the new time zone.

`adjustmentRules` TimeZoneInfo.AdjustmentRule[]

An array of TimeZoneInfo.AdjustmentRule objects that augment the base UTC offset for a particular period.

`disableDaylightSavingTime` Boolean

`true` to discard any daylight saving time-related information present in `adjustmentRules` with the new object; otherwise, `false`.

Returns

TimeZoneInfo

The new time zone. If the `disableDaylightSavingTime` parameter is `true`, the returned object has no daylight saving time data.

Exceptions

ArgumentNullException

The `id` parameter is `null`.

ArgumentException

The `id` parameter is an empty string ("").

-or-

The `baseUtcOffset` parameter does not represent a whole number of minutes.

ArgumentOutOfRangeException

The `baseUtcOffset` parameter is greater than 14 hours or less than -14 hours.

InvalidTimeZoneException

The adjustment rules specified in the `adjustmentRules` parameter overlap.

-or-

The adjustment rules specified in the `adjustmentRules` parameter are not in chronological order.

-or-

One or more elements in `adjustmentRules` are `null`.

-or-

A date can have multiple adjustment rules applied to it.

-or-

The sum of the `baseUtcOffset` parameter and the DaylightDelta value of one or more objects in the `adjustmentRules` array is greater than 14 hours or less than -14 hours.

Examples

The following example creates a custom time zone for the Palmer station and Anvers Island in Antarctica. It sets the `disableDaylightSavingTime` parameter in the call to the TimeZoneInfo.CreateCustomTimeZone method to `true`. It then displays the new time zone's daylight saving time name, if one is present, and the number of adjustment rules to confirm that the new time zone has no daylight saving time information.

C#

```
// Define transition times to/from DST
TimeZoneInfo.TransitionTime startTransition, endTransition;
startTransition = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 4, 0, 0),
                                                                    10, 2,
DayOfWeek.Sunday);
endTransition = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1,3, 0, 0),
                                                                  3, 2,
DayOfWeek.Sunday);
// Define adjustment rule
TimeSpan delta = new TimeSpan(1, 0, 0);
TimeZoneInfo.AdjustmentRule adjustment =
TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new DateTime(1999, 10,
1),
                                        DateTime.MaxValue.Date, delta,
startTransition, endTransition);
// Create array for adjustment rules
TimeZoneInfo.AdjustmentRule[] adjustments = {adjustment};
// Define other custom time zone arguments
string displayName = "(GMT-04:00) Antarctica/Palmer Time";
string standardName = "Palmer Standard Time";
string daylightName = "Palmer Daylight Time";
TimeSpan offset = new TimeSpan(-4, 0, 0);
// Create custom time zone without copying DST information
TimeZoneInfo palmer = TimeZoneInfo.CreateCustomTimeZone(standardName,
offset, displayName, standardName,
                                                    daylightName,
adjustments, true);
// Indicate whether new time zone//s adjustment rules are present
Console.WriteLine("{0} {1}has {2} adjustment rules.",
                  palmer.StandardName,
                  ! (string.IsNullOrEmpty(palmer.DaylightName)) ?  "(" +
palmer.DaylightName + ") ": "" ,
                  palmer.GetAdjustmentRules().Length);
// Indicate whether new time zone supports DST
Console.WriteLine("{0} supports DST: {1}", palmer.StandardName,
palmer.SupportsDaylightSavingTime);
```

Remarks

You can use this overload of the TimeZoneInfo.CreateCustomTimeZone method to create a custom time zone whose support for daylight saving time can be determined by conditions at run time.

The following table shows the relationship between the parameters that are provided to the TimeZoneInfo.CreateCustomTimeZone method and the members of the TimeZoneInfo object that are returned by the method call.

Expand table

CreateCustomTimeZone parameter	TimeZoneInfo property
`id`	Id
`baseUtcOffset`	BaseUtcOffset
`displayName`	DisplayName
`standardDisplayName`	StandardName
`daylightDisplayName`	DaylightName if `disableDaylightSavingTime` is `false`; String.Empty if `disableDaylightSavingTime` is `true`.
`adjustmentRules`	An array of TimeZoneInfo.AdjustmentRule objects returned by the GetAdjustmentRules method if `disableDaylightSavingTime` is `false`; an empty array returned by the GetAdjustmentRules method if `disableDaylightSavingTime` is `true`.
`disableDaylightSavingTime`	Not SupportsDaylightSavingTime.

Typically, the time zone's standard time name and its identifier are the same. However, the length of the time zone's identifier should not exceed 32 characters. The string passed to the `displayName` parameter follows a fairly standard format. The first portion of the display name is the time zone's base offset from Coordinated Universal Time, which is indicated by the acronym GMT (for Greenwich Mean Time), enclosed in parentheses. This is followed by a string that identifies the time zone itself, or one or more of the cities, regions, or countries in the time zone, or both. For example:

```
(GMT+02:00) Athens, Beirut, Istanbul, Minsk
(GMT-02:00) Mid-Atlantic
(GMT-07:00) Mountain Time (US & Canada)
```

A time zone's adjustment rules are defined by doing the following:

1. Calling either the CreateFloatingDateRule or the CreateFixedDateRule method to define the starting and ending transition rules for each adjustment rule.
2. Calling the CreateAdjustmentRule method for each adjustment rule.
3. Assigning the adjustment rules to an array that can be passed as the `adjustmentRules` parameter.

If `disableDaylightSavingTime` parameter is `false`, the operation of this method is identical to the TimeZoneInfo.CreateCustomTimeZone overload. If `disableDaylightSavingTime` is `true`, the returned object includes no adjustment rules and a DaylightName property whose value is an empty string.

See also

- How to: Create Time Zones Without Adjustment Rules
- How to: Create Time Zones with Adjustment Rules

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateCustomTimeZone(String, TimeSpan, String, String)

Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, and a standard time display name.

C#

```
public static TimeZoneInfo CreateCustomTimeZone (string id, TimeSpan
baseUtcOffset, string? displayName, string? standardDisplayName);
```

Parameters

`id` String

The time zone's identifier.

`baseUtcOffset` TimeSpan

An object that represents the time difference between this time zone and Coordinated Universal Time (UTC).

`displayName` String

The display name of the new time zone.

`standardDisplayName` String

The name of the new time zone's standard time.

Returns

TimeZoneInfo

The new time zone.

Exceptions

ArgumentNullException

The `id` parameter is `null`.

ArgumentException

The `id` parameter is an empty string ("").

-or-

The `baseUtcOffset` parameter does not represent a whole number of minutes.

ArgumentOutOfRangeException

The `baseUtcOffset` parameter is greater than 14 hours or less than -14 hours.

Examples

The following example creates a custom time zone for the Mawson and Holme Bay regions of Antarctica. It then displays the result of converting the local time to the time in the new time zone.

```C#
string displayName = "(GMT+06:00) Antarctica/Mawson Time";
string standardName = "Mawson Time";
TimeSpan offset = new TimeSpan(06, 00, 00);
TimeZoneInfo mawson = TimeZoneInfo.CreateCustomTimeZone(standardName,
offset, displayName, standardName);
Console.WriteLine("The current time is {0} {1}",
                  TimeZoneInfo.ConvertTime(DateTime.Now,
TimeZoneInfo.Local, mawson),
                  mawson.StandardName);
```

Remarks

This overload of the CreateCustomTimeZone(String, TimeSpan, String, String) method is suitable for creating a time zone that has no adjustments (that is, a time zone that does not support daylight saving time). To define a time zone that includes adjustments for daylight saving time, use either the TimeZoneInfo.CreateCustomTimeZone or the TimeZoneInfo.CreateCustomTimeZone method.

The following table shows the relationship between the parameters that are provided to the TimeZoneInfo.CreateCustomTimeZone method and the properties of the TimeZoneInfo object that are returned by the method call.

Expand table

CreateCustomTimeZone parameter	TimeZoneInfo property
`id`	Id
`baseUtcOffset`	BaseUtcOffset
`displayName`	DisplayName
`standardDisplayName`	StandardName

Typically, the time zone's standard time name and its identifier are the same. However, the length of the time zone's identifier should not exceed 32 characters. The string passed to the `displayName` parameter follows a fairly standard format. The first portion of the display name is the time zone's base offset from Coordinated Universal Time, which is indicated by the acronym GMT (for Greenwich Mean Time), enclosed in parentheses. This is followed by a string that identifies the time zone itself, or one or more of the cities, regions, or countries in the time zone, or both. For example:

```
(GMT+02:00) Athens, Beirut, Istanbul, Minsk
(GMT-02:00) Mid-Atlantic
(GMT-07:00) Mountain Time (US & Canada)
```

See also

- How to: Create Time Zones Without Adjustment Rules

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[])

Creates a custom time zone with a specified identifier, an offset from Coordinated Universal Time (UTC), a display name, a standard time name, a daylight saving time name, and daylight saving time rules.

C#

```
public static TimeZoneInfo CreateCustomTimeZone (string id, TimeSpan
baseUtcOffset, string? displayName, string? standardDisplayName, string?
daylightDisplayName, TimeZoneInfo.AdjustmentRule[]? adjustmentRules);
```

Parameters

`id` String

The time zone's identifier.

`baseUtcOffset` TimeSpan

An object that represents the time difference between this time zone and Coordinated Universal Time (UTC).

`displayName` String

The display name of the new time zone.

`standardDisplayName` String

The new time zone's standard time name.

`daylightDisplayName` String

The daylight saving time name of the new time zone.

`adjustmentRules` TimeZoneInfo.AdjustmentRule[]

An array that augments the base UTC offset for a particular period.

Returns

TimeZoneInfo

A TimeZoneInfo object that represents the new time zone.

Exceptions

ArgumentNullException

The `id` parameter is `null`.

ArgumentException

The `id` parameter is an empty string ("").

-or-

The `baseUtcOffset` parameter does not represent a whole number of minutes.

ArgumentOutOfRangeException

The `baseUtcOffset` parameter is greater than 14 hours or less than -14 hours.

InvalidTimeZoneException

The adjustment rules specified in the `adjustmentRules` parameter overlap.

-or-

The adjustment rules specified in the `adjustmentRules` parameter are not in chronological order.

-or-

One or more elements in `adjustmentRules` are `null`.

-or-

A date can have multiple adjustment rules applied to it.

-or-

The sum of the `baseUtcOffset` parameter and the DaylightDelta value of one or more objects in the `adjustmentRules` array is greater than 14 hours or less than -14 hours.

Examples

The following example creates a custom time zone for the Palmer station and Anvers Island in Antarctica. It then converts the local time to the time in the new time zone and displays the result.

C#

```
// Define transition times to/from DST
TimeZoneInfo.TransitionTime startTransition, endTransition;
startTransition = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 4, 0, 0),
                                                                    10, 2,
DayOfWeek.Sunday);
endTransition = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 3, 0, 0),
                                                                  3, 2,
DayOfWeek.Sunday);
// Define adjustment rule
TimeSpan delta = new TimeSpan(1, 0, 0);
TimeZoneInfo.AdjustmentRule adjustment;
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1999, 10, 1), DateTime.MaxValue.Date, delta, startTransition,
endTransition);
// Create array for adjustment rules
TimeZoneInfo.AdjustmentRule[] adjustments = {adjustment};
// Define other custom time zone arguments
string displayName = "(GMT-04:00) Antarctica/Palmer Time";
string standardName = "Palmer Time";
string daylightName = "Palmer Daylight Time";
TimeSpan offset = new TimeSpan(-4, 0, 0);
TimeZoneInfo palmer = TimeZoneInfo.CreateCustomTimeZone(standardName,
```

```
offset, displayName, standardName, daylightName, adjustments);
Console.WriteLine("The current time is {0} {1}",
                  TimeZoneInfo.ConvertTime(DateTime.Now,
TimeZoneInfo.Local, palmer),
                  palmer.StandardName);
```

Remarks

This overload of the TimeZoneInfo.CreateCustomTimeZone method is suitable for creating a time zone that supports daylight saving time. To define a time zone that does not support daylight saving time, use either the TimeZoneInfo.CreateCustomTimeZone or the TimeZoneInfo.CreateCustomTimeZone method.

The following table shows the relationship between the parameters that are provided to the TimeZoneInfo.CreateCustomTimeZone method and the members of the TimeZoneInfo object that are returned by the method call.

Expand table

CreateCustomTimeZone parameter	TimeZoneInfo member
`id`	Id
`baseUtcOffset`	BaseUtcOffset
`displayName`	DisplayName
`standardDisplayName`	StandardName
`daylightDisplayName`	DaylightName
`adjustmentRules`	An array of System.TimeZoneInfo.AdjustmentRule objects returned by the GetAdjustmentRules method.

Typically, the time zone's standard time name and its identifier are the same. However, the length of the time zone's identifier should not exceed 32 characters. The string passed to the `displayName` parameter follows a fairly standard format. The first portion of the display name is the time zone's base offset from Coordinated Universal Time, which is indicated by the acronym GMT (for Greenwich Mean Time), enclosed in parentheses. This is followed by a string that identifies the time zone itself, or one or more of the cities, regions, or countries in the time zone, or both. For example:

```
(GMT+02:00) Athens, Beirut, Istanbul, Minsk
(GMT-02:00) Mid-Atlantic
(GMT-07:00) Mountain Time (US & Canada)
```

The `baseUtcOffset` parameter defines the custom time zone's offset from Coordinated Universal Time (UTC) for all of the time zone's adjustment rules. In other words, the TimeZoneInfo object model assumes that the time zone's offset from UTC is constant throughout the life of the time zone, and exists independent of particular adjustment rules. To reflect a time zone that has changed its offset from UTC, you must create a new time zone object.

A time zone's adjustment rules are defined by doing the following:

1. Calling either the CreateFloatingDateRule or the CreateFixedDateRule method to define the starting and ending transition time for each adjustment rule.
2. Calling the CreateAdjustmentRule method for each adjustment rule.
3. Assigning the adjustment rules to an array that can be passed as the `adjustmentRules` parameter.

See also

- How to: Create Time Zones with Adjustment Rules

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeZoneInfo.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two TimeZoneInfo objects are equal.

Overloads

Expand table

Equals(Object)	Determines whether the current TimeZoneInfo object and another object are equal.
Equals(TimeZoneInfo)	Determines whether the current TimeZoneInfo object and another TimeZoneInfo object are equal.

Equals(Object)

Determines whether the current TimeZoneInfo object and another object are equal.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

A second object to compare with the current object.

Returns

Boolean

`true` if `obj` is a TimeZoneInfo object that is equal to the current instance; otherwise, `false`.

Examples

The following example uses the Equals(Object) method to determine whether the local time zone is Pacific Time or Eastern Time.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      TimeZoneInfo thisTimeZone;
      object obj1, obj2;

      thisTimeZone = TimeZoneInfo.Local;
      obj1 = TimeZoneInfo.FindSystemTimeZoneById("Pacific Standard
Time");
      obj2 = TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard
Time");
      Console.WriteLine(thisTimeZone.Equals(obj1));
      Console.WriteLine(thisTimeZone.Equals(obj2));
   }
}
// The example displays the following output:
//      True
//      False
```

Remarks

This method converts or casts `obj` to a TimeZoneInfo object and compares it with the current instance. The test for equality is based on a comparison of values. The current TimeZoneInfo instance and `obj` are considered to be equal under the following conditions:

- The run-time type of `obj` is TimeZoneInfo.
- The two objects have the same Id property value.
- The two objects have the same adjustment rules.

If `obj` is `null`, this method returns `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(TimeZoneInfo)

Determines whether the current TimeZoneInfo object and another TimeZoneInfo object are equal.

C#

```
public bool Equals (TimeZoneInfo? other);
```

Parameters

`other` TimeZoneInfo

A second object to compare with the current object.

Returns

Boolean

`true` if the two TimeZoneInfo objects are equal; otherwise, `false`.

Implements

Equals(T)

Examples

The following example uses the Equals(TimeZoneInfo) method to determine whether the local time zone is Pacific Time or Eastern Time.

C#

```
    TimeZoneInfo thisTimeZone, zone1, zone2;

    thisTimeZone = TimeZoneInfo.Local;
    zone1 = TimeZoneInfo.FindSystemTimeZoneById("Pacific Standard Time");
```

```
    zone2 = TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard Time");
    Console.WriteLine(thisTimeZone.Equals(zone1));
    Console.WriteLine(thisTimeZone.Equals(zone2));
```

Remarks

Equality is based on a comparison of values. Two TimeZoneInfo objects are considered to be equal under the following conditions:

- The value of their Id property is the same.
- They have the same adjustment rules.

TimeZoneInfo.Equals(TimeZoneInfo) returns the Boolean value that results from evaluating the following expression:

```C#
other.Id == this.Id && HasSameRules(other);
```

If the `other` parameter is an uninitialized TimeZoneInfo object, this method returns `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TimeZoneInfo.FindSystemTimeZoneById(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Instantiates a new TimeZoneInfo object based on its identifier.

C#

```
public static TimeZoneInfo FindSystemTimeZoneById (string id);
```

Parameters

id String

The time zone identifier, which corresponds to the Id property.

Returns

TimeZoneInfo

An object whose identifier is the value of the `id` parameter.

Exceptions

OutOfMemoryException

The system does not have enough memory to hold information about the time zone.

ArgumentNullException

The `id` parameter is `null`.

TimeZoneNotFoundException

The time zone identifier specified by `id` was not found. This means that a time zone identifier whose name matches `id` does not exist, or that the identifier exists but does not contain any time zone data.

SecurityException

The process does not have the permissions required to read from the registry key that contains the time zone information.

InvalidTimeZoneException

The time zone identifier was found, but the registry data is corrupted.

Examples

The following example uses the FindSystemTimeZoneById method to retrieve the Tokyo Standard Time zone. This TimeZoneInfo object is then used to convert the local time to the time in Tokyo and to determine whether it is Tokyo Standard Time or Tokyo Daylight Time.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get time in local time zone
      DateTime thisTime = DateTime.Now;
      Console.WriteLine("Time in {0} zone: {1}",
TimeZoneInfo.Local.IsDaylightSavingTime(thisTime) ?
                        TimeZoneInfo.Local.DaylightName :
TimeZoneInfo.Local.StandardName, thisTime);
      Console.WriteLine("   UTC Time: {0}",
TimeZoneInfo.ConvertTimeToUtc(thisTime, TimeZoneInfo.Local));
      // Get Tokyo Standard Time zone
      TimeZoneInfo tst = TimeZoneInfo.FindSystemTimeZoneById("Tokyo Standard
Time");
      DateTime tstTime = TimeZoneInfo.ConvertTime(thisTime,
TimeZoneInfo.Local, tst);
      Console.WriteLine("Time in {0} zone: {1}",
tst.IsDaylightSavingTime(tstTime) ?
                        tst.DaylightName : tst.StandardName, tstTime);
      Console.WriteLine("   UTC Time: {0}",
TimeZoneInfo.ConvertTimeToUtc(tstTime, tst));
   }
}
// The example displays output like the following when run on a system in
the
// U.S. Pacific Standard Time zone:
//       Time in Pacific Standard Time zone: 12/6/2013 10:57:51 AM
//          UTC Time: 12/6/2013 6:57:51 PM
```

```
//       Time in Tokyo Standard Time zone: 12/7/2013 3:57:51 AM
//          UTC Time: 12/6/2013 6:57:51 PM
```

Remarks

The `id` parameter must correspond exactly to the time zone's identifier in length, but not in case, for a successful match to occur; that is, the comparison of `id` with time zone identifiers is case-insensitive. If you want to retrieve time zone objects based on partial matches, you can write custom procedures that work with the read-only collection of TimeZoneInfo objects returned by the GetSystemTimeZones method.

On Windows systems, `FindSystemTimeZoneById` tries to match `id` to the subkey names of the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Time Zones branch of the registry. On Linux and macOS, it uses time zone information available in the ICU Library. If the registry or the library does not have the information for the time zone you desire, you can create a particular time zone either by calling one of the overloads of the CreateCustomTimeZone method or by calling FromSerializedString to deserialize a TimeZoneInfo object that represents the required time zone. However, time zones created by these method calls are not system-defined time and cannot be retrieved using the FindSystemTimeZoneById method. These custom time zones can be accessed only through the object reference returned by the CreateCustomTimeZone or FromSerializedString method call.

This method returns a new TimeZoneInfo instance for each method call; it does not return cached objects. This may impact performance in applications that call the `FindSystemTimeZoneById` method repeatedly with the same identifier.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Id
- How to: Instantiate a TimeZoneInfo Object

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.FromSerializedString(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Deserializes a string to re-create an original serialized TimeZoneInfo object.

```
C#
public static TimeZoneInfo FromSerializedString (string source);
```

Parameters

`source` String

The string representation of the serialized TimeZoneInfo object.

Returns

TimeZoneInfo

The original serialized object.

Exceptions

ArgumentException

The `source` parameter is Empty.

ArgumentNullException

The `source` parameter is a null string.

SerializationException

The source parameter cannot be deserialized back into a TimeZoneInfo object.

Examples

The following example tries to retrieve the Antarctica/South Pole time zone from the local system. If it fails, the code tries to retrieve information about the time zone from a text file in the application directory. If this attempt fails, the code creates the time zone and writes information about the time zone to the text file.

C#

```
private TimeZoneInfo InitializeTimeZone()
{
   TimeZoneInfo southPole = null;
   // Determine if South Pole time zone is defined in system
   try
   {
      southPole = TimeZoneInfo.FindSystemTimeZoneById("Antarctica/South Pole
Standard Time");
   }
   // Time zone does not exist; create it, store it in a text file, and
return it
   catch
   {
      const string filename = @".\TimeZoneInfo.txt";
      bool found = false;

      if (File.Exists(filename))
      {
         StreamReader reader = new StreamReader(filename);
         string timeZoneInfo;
         while (reader.Peek() >= 0)
         {
            timeZoneInfo = reader.ReadLine();
            if (timeZoneInfo.Contains("Antarctica/South Pole"))
            {
               southPole = TimeZoneInfo.FromSerializedString(timeZoneInfo);
               reader.Close();
               found = true;
               break;
            }
         }
      }
      if (! found)
      {
         // Define transition times to/from DST
         TimeZoneInfo.TransitionTime startTransition =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1, 2,
0, 0), 10, 1, DayOfWeek.Sunday);
         TimeZoneInfo.TransitionTime endTransition =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1, 2,
0, 0), 3, 3, DayOfWeek.Sunday);
         // Define adjustment rule
         TimeSpan delta = new TimeSpan(1, 0, 0);
         TimeZoneInfo.AdjustmentRule adjustment =
TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new DateTime(1989, 10, 1),
DateTime.MaxValue.Date, delta, startTransition, endTransition);
```

```
            // Create array for adjustment rules
            TimeZoneInfo.AdjustmentRule[] adjustments = {adjustment};
            // Define other custom time zone arguments
            string displayName = "(GMT+12:00) Antarctica/South Pole";
            string standardName = "Antarctica/South Pole Standard Time";
            string daylightName = "Antarctica/South Pole Daylight Time";
            TimeSpan offset = new TimeSpan(12, 0, 0);
            southPole = TimeZoneInfo.CreateCustomTimeZone(standardName, offset,
displayName, standardName, daylightName, adjustments);
            // Write time zone to the file
            StreamWriter writer = new StreamWriter(filename, true);
            writer.WriteLine(southPole.ToSerializedString());
            writer.Close();
        }
    }
    return southPole;
}
```

Remarks

There is an alternative to providing all the code required to create a time zone that is not found in the registry on Windows or in the ICU Library on Linux and macOS. You can define a custom time zone and either use the ToSerializedString method in a standalone executable or use an application's setup program to save the time zone as a string. The application can then retrieve this string from its storage location and instantiate it using the FromSerializedString method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ToSerializedString()
- How to: Restore Time Zones from an Embedded Resource

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.GetAdjustmentRules Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves an array of TimeZoneInfo.AdjustmentRule objects that apply to the current TimeZoneInfo object.

C#

```
public TimeZoneInfo.AdjustmentRule[] GetAdjustmentRules ();
```

Returns

AdjustmentRule[]

An array of objects for this time zone.

Exceptions

OutOfMemoryException

The system does not have enough memory to make an in-memory copy of the adjustment rules.

Examples

The following example retrieves all time zones that are defined on the local system and displays complete information about their adjustment rules to the console.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
```

```
   Last = 5,
}

private static void ShowStartAndEndDates()
{
   // Get all time zones from system
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   string[] monthNames =
CultureInfo.CurrentCulture.DateTimeFormat.MonthNames;
   // Get each time zone
   foreach (TimeZoneInfo timeZone in timeZones)
   {
      TimeZoneInfo.AdjustmentRule[] adjustments =
timeZone.GetAdjustmentRules();
      // Display message for time zones with no adjustments
      if (adjustments.Length == 0)
      {
         Console.WriteLine("{0} has no adjustment rules",
timeZone.StandardName);
      }
      else
      {
         // Handle time zones with 1 or 2+ adjustments differently
         bool showCount = false;
         int ctr = 0;
         string spacer = "";

         Console.WriteLine("{0} Adjustment rules", timeZone.StandardName);
         if (adjustments.Length > 1)
         {
            showCount = true;
            spacer = "   ";
         }
         // Iterate adjustment rules
         foreach (TimeZoneInfo.AdjustmentRule adjustment in adjustments)
         {
            if (showCount)
            {
               Console.WriteLine("   Adjustment rule #{0}", ctr+1);
               ctr++;
            }
            // Display general adjustment information
            Console.WriteLine("{0}   Start Date: {1:D}", spacer,
adjustment.DateStart);
            Console.WriteLine("{0}   End Date: {1:D}", spacer,
adjustment.DateEnd);
            Console.WriteLine("{0}   Time Change: {1}:{2:00} hours", spacer,
                              adjustment.DaylightDelta.Hours,
adjustment.DaylightDelta.Minutes);
            // Get transition start information
            TimeZoneInfo.TransitionTime transitionStart =
adjustment.DaylightTransitionStart;
            Console.Write("{0}   Annual Start: ", spacer);
            if (transitionStart.IsFixedDateRule)
```

```
            {
               Console.WriteLine("On {0} {1} at {2:t}",
                                 monthNames[transitionStart.Month - 1],
                                 transitionStart.Day,
                                 transitionStart.TimeOfDay);
            }
            else
            {
               Console.WriteLine("The {0} {1} of {2} at {3:t}",

((WeekOfMonth)transitionStart.Week).ToString(),
                                 transitionStart.DayOfWeek.ToString(),
                                 monthNames[transitionStart.Month - 1],
                                 transitionStart.TimeOfDay);
            }
            // Get transition end information
            TimeZoneInfo.TransitionTime transitionEnd =
adjustment.DaylightTransitionEnd;
            Console.Write("{0}   Annual End: ", spacer);
            if (transitionEnd.IsFixedDateRule)
            {
               Console.WriteLine("On {0} {1} at {2:t}",
                                 monthNames[transitionEnd.Month - 1],
                                 transitionEnd.Day,
                                 transitionEnd.TimeOfDay);
            }
            else
            {
               Console.WriteLine("The {0} {1} of {2} at {3:t}",

((WeekOfMonth)transitionEnd.Week).ToString(),
                                 transitionEnd.DayOfWeek.ToString(),
                                 monthNames[transitionEnd.Month - 1],
                                 transitionEnd.TimeOfDay);
            }
         }
      }
      Console.WriteLine();
   }
}
```

Remarks

The GetAdjustmentRules method retrieves an array of System.TimeZoneInfo.AdjustmentRule objects. Each object in the array defines the effective start and end date of that time zone adjustment, as well as its delta (the exact amount by which the adjustment causes the time to change). In addition, two properties return System.TimeZoneInfo.TransitionTime objects that define when each annual transition to and from standard time occurs.

If a time zone has multiple adjustment rules, they are generally ordered from earliest (at index 0) to latest (at index Length - 1).

If a time zone has no adjustment rules, the GetAdjustmentRules method returns an empty array (an array whose Length is zero).

Any modifications to the elements of the array returned by the GetAdjustmentRules method are not reflected in the adjustment rules that belong to a particular time zone. To modify a time zone's adjustment rules (such as to reflect its historical transition to and from daylight saving time) you must create a new time zone with the appropriate adjustment rules, rather than modify the existing one.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TimeZoneInfo.AdjustmentRule

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.GetAmbiguousTime Offsets Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns information about the possible dates and times that an ambiguous date and time can be mapped to.

Overloads

Expand table

GetAmbiguousTimeOffsets(DateTime)	Returns information about the possible dates and times that an ambiguous date and time can be mapped to.
GetAmbiguousTimeOffsets(DateTimeOffset)	Returns information about the possible dates and times that an ambiguous date and time can be mapped to.

GetAmbiguousTimeOffsets(DateTime)

Returns information about the possible dates and times that an ambiguous date and time can be mapped to.

C#

```
public TimeSpan[] GetAmbiguousTimeOffsets (DateTime dateTime);
```

Parameters

`dateTime` DateTime

A date and time.

Returns

TimeSpan[]

An array of objects that represents possible Coordinated Universal Time (UTC) offsets that a particular date and time can be mapped to.

Exceptions

ArgumentException

`dateTime` is not an ambiguous time.

Examples

The following example defines a method named `ShowPossibleUtcTimes` that uses the GetAmbiguousTimeOffsets(DateTime) method to map an ambiguous time to its possible corresponding Coordinated Universal Time (UTC) times.

C#

```
private void ShowPossibleUtcTimes(DateTime ambiguousTime, TimeZoneInfo
timeZone)
{
   // Determine if time is ambiguous in target time zone
   if (! timeZone.IsAmbiguousTime(ambiguousTime))
   {
      Console.WriteLine("{0} is not ambiguous in time zone {1}.",
                        ambiguousTime,
                        timeZone.DisplayName);
   }
   else
   {
      // Display time and its time zone (local, UTC, or indicated by
timeZone argument)
      string originalTimeZoneName;
      if (ambiguousTime.Kind == DateTimeKind.Utc)
         originalTimeZoneName = "UTC";
      else if (ambiguousTime.Kind == DateTimeKind.Local)
         originalTimeZoneName = "local time";
      else
         originalTimeZoneName = timeZone.DisplayName;

      Console.WriteLine("{0} {1} maps to the following possible times:",
                        ambiguousTime, originalTimeZoneName);
      // Get ambiguous offsets
      TimeSpan[] offsets =
timeZone.GetAmbiguousTimeOffsets(ambiguousTime);
      // Handle times not in time zone of timeZone argument
      // Local time where timeZone is not local zone
      if ((ambiguousTime.Kind == DateTimeKind.Local) && !
```

```
timeZone.Equals(TimeZoneInfo.Local))
         ambiguousTime = TimeZoneInfo.ConvertTime(ambiguousTime,
TimeZoneInfo.Local, timeZone);
      // UTC time where timeZone is not UTC zone
      else if ((ambiguousTime.Kind == DateTimeKind.Utc) && !
timeZone.Equals(TimeZoneInfo.Utc))
         ambiguousTime = TimeZoneInfo.ConvertTime(ambiguousTime,
TimeZoneInfo.Utc, timeZone);

      // Display each offset and its mapping to UTC
      foreach (TimeSpan offset in offsets)
      {
         if (offset.Equals(timeZone.BaseUtcOffset))
            Console.WriteLine("If {0} is {1}, {2} UTC", ambiguousTime,
timeZone.StandardName, ambiguousTime - offset);
         else
            Console.WriteLine("If {0} is {1}, {2} UTC", ambiguousTime,
timeZone.DaylightName, ambiguousTime - offset);
      }
   }
}
```

The method can then be called using code such as the following:

C#

```
Console.WriteLine();
ShowPossibleUtcTimes(new DateTime(2007, 11, 4, 1, 0, 0),
                     TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time"));
Console.WriteLine();
ShowPossibleUtcTimes(new DateTime(2007, 11, 4, 01, 00, 00,
DateTimeKind.Local),
                     TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time"));
Console.WriteLine();
ShowPossibleUtcTimes(new DateTime(2007, 11, 4, 00, 00, 00,
DateTimeKind.Local),
                     TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time"));
Console.WriteLine();
ShowPossibleUtcTimes(new DateTime(2007, 11, 4, 01, 00, 00,
DateTimeKind.Unspecified),
                     TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time"));
Console.WriteLine();
ShowPossibleUtcTimes(new DateTime(2007, 11, 4, 07, 00, 00,
DateTimeKind.Utc),
                     TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time"));
//
// This example produces the following output if run in the Pacific time
zone:
```

```
//
//    11/4/2007 1:00:00 AM (GMT-06:00) Central Time (US & Canada) maps to
the following possible times:
//    If 11/4/2007 1:00:00 AM is Central Standard Time, 11/4/2007 7:00:00
AM UTC
//    If 11/4/2007 1:00:00 AM is Central Daylight Time, 11/4/2007 6:00:00
AM UTC
//
//    11/4/2007 1:00:00 AM Pacific Standard Time is not ambiguous in time
zone (GMT-06:00) Central Time (US & Canada).
//
//    11/4/2007 12:00:00 AM local time maps to the following possible
times:
//    If 11/4/2007 1:00:00 AM is Central Standard Time, 11/4/2007 7:00:00
AM UTC
//    If 11/4/2007 1:00:00 AM is Central Daylight Time, 11/4/2007 6:00:00
AM UTC
//
//    11/4/2007 1:00:00 AM (GMT-06:00) Central Time (US & Canada) maps to
the following possible times:
//    If 11/4/2007 1:00:00 AM is Central Standard Time, 11/4/2007 7:00:00
AM UTC
//    If 11/4/2007 1:00:00 AM is Central Daylight Time, 11/4/2007 6:00:00
AM UTC
//
//    11/4/2007 7:00:00 AM UTC maps to the following possible times:
//    If 11/4/2007 1:00:00 AM is Central Standard Time, 11/4/2007 7:00:00
AM UTC
//    If 11/4/2007 1:00:00 AM is Central Daylight Time, 11/4/2007 6:00:00
AM UTC
//
```

Remarks

The precise behavior of this method depends on the relationship between the Kind property and the TimeZoneInfo object, as the following table shows.

Expand table

TimeZoneInfo object type	Kind property value	Behavior
TimeZoneInfo.Local	DateTimeKind.Local or DateTimeKind.Unspecified	Returns ambiguous time offsets for `dateTime`.
TimeZoneInfo.Local	DateTimeKind.Utc	Converts `dateTime` to the local time, and then returns ambiguous time offsets for that time.
TimeZoneInfo.Utc	Any value.	Throws an ArgumentException.

TimeZoneInfo object type	Kind property value	Behavior
Any other time zone.	Local or DateTimeKind.Utc	Converts `dateTime` to the specified time zone, and then determines whether that time is ambiguous.
Any other time zone.	DateTimeKind.Unspecified	Determines whether `dateTime` is ambiguous in the specified time zone.

The order of TimeSpan objects in the array returned by this method is undefined. However, you can determine which element represents an offset from the time zone's standard time by comparing its value with the time zone's BaseUtcOffset property. To map an ambiguous time to a time zone's standard time, see How to: Resolve Ambiguous Times.

See also

- IsAmbiguousTime
- How to: Resolve Ambiguous Times
- How to: Let Users Resolve Ambiguous Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetAmbiguousTimeOffsets(DateTimeOffset)

Returns information about the possible dates and times that an ambiguous date and time can be mapped to.

```C#
public TimeSpan[] GetAmbiguousTimeOffsets (DateTimeOffset
dateTimeOffset);
```

Parameters

`dateTimeOffset` DateTimeOffset

A date and time.

Returns

TimeSpan[]

An array of objects that represents possible Coordinated Universal Time (UTC) offsets that a particular date and time can be mapped to.

Exceptions

ArgumentException

`dateTimeOffset` is not an ambiguous time.

Remarks

The precise behavior of this method depends on the relationship between the Offset property of the `dateTimeOffset` parameter and the TimeZoneInfo object. If the value of the Offset property corresponds to the current time zone's possible offsets from Coordinated Universal Time (UTC) for that date and time, the method returns the possible offsets. Otherwise, it converts `dateTimeOffset` to the time in the current time zone, and then returns the possible offsets of that date and time.

The order of TimeSpan objects in the array returned by this method is undefined. However, you can determine which element represents an offset from the time zone's standard time by comparing its value with the time zone's BaseUtcOffset property. To map an ambiguous time to a time zone's standard time, see How to: Resolve Ambiguous Times.

See also

- IsAmbiguousTime(DateTime)
- How to: Resolve Ambiguous Times
- How to: Let Users Resolve Ambiguous Times

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as a hash function for hashing algorithms and data structures such as hash tables.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that serves as the hash code for this TimeZoneInfo object.

Remarks

For additional detail about the GetHashCode method, see GetHashCode.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.GetSystemTimeZones Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetSystemTimeZones()	Returns a sorted collection of all the time zones about which information is available on the local system.
GetSystemTimeZones(Boolean)	Returns a ReadOnlyCollection<T> containing all valid TimeZone's from the local machine. This method does *not* throw TimeZoneNotFoundException or InvalidTimeZoneException.

GetSystemTimeZones()

Returns a sorted collection of all the time zones about which information is available on the local system.

C#

```
public static
System.Collections.ObjectModel.ReadOnlyCollection<TimeZoneInfo>
GetSystemTimeZones ();
```

Returns

ReadOnlyCollection<TimeZoneInfo>

A read-only collection of TimeZoneInfo objects.

Exceptions

OutOfMemoryException

There is insufficient memory to store all time zone information.

SecurityException

The user does not have permission to read from the registry keys that contain time zone information.

Examples

The following example retrieves a collection of time zone objects that represent the time zones defined on a computer and writes information about them to a text file.

C#

```
using System;
using System.Globalization;
using System.IO;
using System.Collections.ObjectModel;

public class Example
{
   public static void Main()
   {
      const string OUTPUTFILENAME = @"C:\Temp\TimeZoneInfo.txt";

      DateTimeFormatInfo dateFormats =
CultureInfo.CurrentCulture.DateTimeFormat;
      ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
      StreamWriter sw = new StreamWriter(OUTPUTFILENAME, false);

      foreach (TimeZoneInfo timeZone in timeZones)
      {
         bool hasDST = timeZone.SupportsDaylightSavingTime;
         TimeSpan offsetFromUtc = timeZone.BaseUtcOffset;
         TimeZoneInfo.AdjustmentRule[] adjustRules;
         string offsetString;

         sw.WriteLine("ID: {0}", timeZone.Id);
         sw.WriteLine("   Display Name: {0, 40}", timeZone.DisplayName);
         sw.WriteLine("   Standard Name: {0, 39}",
timeZone.StandardName);
         sw.Write("   Daylight Name: {0, 39}", timeZone.DaylightName);
         sw.Write(hasDST ? "   ***Has " : "   ***Does Not Have ");
         sw.WriteLine("Daylight Saving Time***");
         offsetString = String.Format("{0} hours, {1} minutes",
offsetFromUtc.Hours, offsetFromUtc.Minutes);
         sw.WriteLine("   Offset from UTC: {0, 40}", offsetString);
         adjustRules = timeZone.GetAdjustmentRules();
         sw.WriteLine("   Number of adjustment rules: {0, 26}",
```

```
adjustRules.Length);
         if (adjustRules.Length > 0)
         {
            sw.WriteLine("   Adjustment Rules:");
            foreach (TimeZoneInfo.AdjustmentRule rule in adjustRules)
            {
               TimeZoneInfo.TransitionTime transTimeStart =
rule.DaylightTransitionStart;
               TimeZoneInfo.TransitionTime transTimeEnd =
rule.DaylightTransitionEnd;

               sw.WriteLine("      From {0} to {1}", rule.DateStart,
rule.DateEnd);
               sw.WriteLine("      Delta: {0}", rule.DaylightDelta);
               if (! transTimeStart.IsFixedDateRule)
               {
                  sw.WriteLine("      Begins at {0:t} on {1} of week {2}
of {3}", transTimeStart.TimeOfDay,

transTimeStart.DayOfWeek,

transTimeStart.Week,

dateFormats.MonthNames[transTimeStart.Month - 1]);
                  sw.WriteLine("      Ends at {0:t} on {1} of week {2} of
{3}", transTimeEnd.TimeOfDay,

transTimeEnd.DayOfWeek,

transTimeEnd.Week,

dateFormats.MonthNames[transTimeEnd.Month - 1]);
               }
               else
               {
                  sw.WriteLine("      Begins at {0:t} on {1} {2}",
transTimeStart.TimeOfDay,

transTimeStart.Day,

dateFormats.MonthNames[transTimeStart.Month - 1]);
                  sw.WriteLine("      Ends at {0:t} on {1} {2}",
transTimeEnd.TimeOfDay,

transTimeEnd.Day,

dateFormats.MonthNames[transTimeEnd.Month - 1]);
               }
            }
         }
      }
      sw.Close();
   }
}
```

PowerShell

```
#  Get timezone/date details and open a stream writer
$DateFormats    =
[System.Globalization.CultureInfo]::CurrentCulture.DateTimeFormat
$TimeZones      = [System.TimeZoneInfo]::GetSystemTimeZones()
$OutputFileName = 'C:\Temp\TimeZoneInfo.txt'
$Sw             = New-Object -TypeName System.IO.StreamWriter -
ArgumentList $OutputFileName,$false
# Write overview information
$Sw.WriteLine('{0} Time zones on this system' -f $TimeZones.Count)

# Process each timezone on the system
# Write details to the streamwriter
foreach ($TimeZone in $TimeZones) {
 $HasDST        = $TimeZone.SupportsDaylightSavingTime
 $OffsetFromUtc = $TimeZone.BaseUtcOffset
 $Sw.WriteLine("ID: {0}" -f $TimeZone.Id)
 $Sw.WriteLine("   Display Name:  {0}" -f $TimeZone.DisplayName)
 $Sw.WriteLine("   Standard Name: {0}" -f $TimeZone.StandardName)
 $Sw.Write("   Daylight Name: {0}" -f $TimeZone.DaylightName)
 $Sw.Write( $(If ($HasDST) {"  ***Has: "} Else {"  ***Does Not Have: "}))
 $Sw.WriteLine("Daylight Saving Time***")
 $OffsetString = "{0} hours, {1} minutes" -f $OffsetFromUtc.Hours,
$OffsetFromUtc.Minutes
 $Sw.WriteLine("   Offset from UTC: {0}"  -f $offsetString)
 $AdjustRules = $timeZone.GetAdjustmentRules()
 $Sw.WriteLine("   Number of adjustment rules: {0}" -f
$adjustRules.Count)
 If ($AdjustRules.Count -gt 0)
 {
    $Sw.WriteLine("   Adjustment Rules:")
    foreach ($Rule in $AdjustRules)
    {
       $TransTimeStart = $Rule.DaylightTransitionStart
       $TransTimeEnd   = $Rule.DaylightTransitionEnd
       $Sw.WriteLine(("     From   {0} to {1}" -f $Rule.DateStart,
$Rule.DateEnd))
       $Sw.WriteLine(("       Delta: {0}" -f $Rule.DaylightDelta))
       if (-Not  $TransTimeStart.IsFixedDateRule)
       {
          $Sw.WriteLine(("       Begins at {0:t} on {1} of week {2} of
{3}" -f $TransTimeStart.TimeOfDay,

$TransTimeStart.DayOfWeek,

$TransTimeStart.Week,

$DateFormats.MonthNames[$TransTimeStart.Month - 1]))
          $Sw.WriteLine(("       Ends at {0:t} on {1} of week {2} of {3}"
-f $TransTimeEnd.TimeOfDay,

$TransTimeEnd.DayOfWeek,
```

```
$TransTimeEnd.Week,

$DateFormats.MonthNames[[int] $TransTimeEnd.Month - 1]))
       }
     else
       {
         $Sw.WriteLine(("       Begins at {0:t} on {1} {2}" -f
$TransTimeStart.TimeOfDay,

$TransTimeStart.Day,

$DateFormats.MonthNames[$transTimeStart.Month - 1]))
         $Sw.WriteLine(("       Ends at {0:t} on {1} {2}"   -f
$TransTimeEnd.TimeOfDay,

$TransTimeEnd.Day,

$DateFormats.MonthNames[$transTimeEnd.Month - 1]))
       }
    } # End of processing each adjustment rule
 } # End of checking adjustment rules
} # End of processing Time Zones

# Close stream writer then display output in notepad
$Sw.Close()

#  Output of 1st three time zones:
#  135 Time zones on this system
#  ID: Dateline Standard Time
#     Display Name:  (UTC-12:00) International Date Line West
#     Standard Name: Dateline Standard Time
#     Daylight Name: Dateline Summer Time  ***Does Not Have: Daylight
Saving Time***
#     Offset from UTC: -12 hours, 0 minutes
#     Number of adjustment rules: 0
#  ID: UTC-11
#     Display Name:  (UTC-11:00) Co-ordinated Universal Time-11
#     Standard Name: UTC-11
#     Daylight Name: UTC-11  ***Does Not Have: Daylight Saving Time***
#     Offset from UTC: -11 hours, 0 minutes
#     Number of adjustment rules: 0
#  ID: Aleutian Standard Time
#     Display Name:  (UTC-10:00) Aleutian Islands
#     Standard Name: Aleutian Standard Time
#     Daylight Name: Aleutian Summer Time  ***Has: Daylight Saving
Time***
#     Offset from UTC: -10 hours, 0 minutes
#     Number of adjustment rules: 2
#     Adjustment Rules:
#       From   01/01/0001 00:00:00 to 31/12/2006 00:00:00
#         Delta: 01:00:00
#         Begins at 02:00 on Sunday of week 1 of April
#         Ends at 02:00 on Sunday of week 5 of October
#       From   01/01/2007 00:00:00 to 31/12/9999 00:00:00
#         Delta: 01:00:00
```

```
#         Begins at 02:00 on Sunday of week 2 of March
#         Ends at 02:00 on Sunday of week 1 of November
```

Remarks

The GetSystemTimeZones method retrieves all available time zone information from the subkeys of the registry's HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Time Zones key on Windows systems and from the ICU Library on Linux and macOS. If it is unable to successfully retrieve and parse values for particular string properties of individual TimeZoneInfo objects, this method sets their value to an empty string ("").

> **Important**
>
> The **GetSystemTimeZones** method returns a collection of **TimeZoneInfo** objects only for time zones defined in the Windows registry or the ICU library. It does not include time zones created using the overloads of the **CreateCustomTimeZone** method. These are accessible only through the object reference returned by the time zone creation method.

The collection returned by this method is sorted by UTC offset and, for time zones that have the same UTC offset, by the display name using the current culture. For information about the display name, see DisplayName.

The ReadOnlyCollection<T> object returned by this method supports the IEnumerable<T> interface, which means that it can be iterated using the `foreach` (in C#) or `For Each…Next` (in Visual Basic) statements. Each iteration of the loop provides the next TimeZoneInfo object in the collection.

The collection of TimeZoneInfo objects represents time zones defined on the local computer; it does not necessarily provide complete information for all time zones during all time periods. If your application requires time zones not found on the local computer, you can create custom time zones using the overloads of the CreateCustomTimeZone method. For more information, see How to: Create Time Zones Without Adjustment Rules and How to: Create Time Zones with Adjustment Rules.

You can also determine whether an individual time zone is defined on the local computer by calling the FindSystemTimeZoneById method and providing the identifier of the time zone you want to retrieve as a parameter.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetSystemTimeZones(Boolean)

Returns a ReadOnlyCollection<T> containing all valid TimeZone's from the local machine. This method does *not* throw TimeZoneNotFoundException or InvalidTimeZoneException.

C#

```
public static
System.Collections.ObjectModel.ReadOnlyCollection<TimeZoneInfo>
GetSystemTimeZones (bool skipSorting);
```

Parameters

`skipSorting` Boolean

If `true`, The collection returned may not necessarily be sorted.

Returns

ReadOnlyCollection<TimeZoneInfo>

Remarks

By setting the skipSorting parameter to `true`, the method will attempt to avoid sorting the returned collection. This option can be beneficial when the caller does not require a sorted list and aims to enhance the performance.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.GetUtcOffset Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.

Overloads

Expand table

GetUtcOffset(DateTime)	Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.
GetUtcOffset(DateTimeOffset)	Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.

GetUtcOffset(DateTime)

Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.

C#

```
public TimeSpan GetUtcOffset (DateTime dateTime);
```

Parameters

`dateTime` DateTime

The date and time to determine the offset for.

Returns

TimeSpan

An object that indicates the time difference between the two time zones.

Examples

The following example illustrates the use of the GetUtcOffset(DateTime) method with different time zones and with date values that have different Kind property values.

C#

```
using System;

[assembly:CLSCompliant(true)]
namespace TimeZoneInfoCode
{
   public class TimeOffsets
   {
      public static void Main()
      {
         TimeOffsets timeoff = new TimeOffsets();
         TimeZoneInfo cst = TimeZoneInfo.FindSystemTimeZoneById("Central
Standard Time");

         timeoff.ShowOffset(new DateTime(2006, 6, 12, 11, 0, 0),
TimeZoneInfo.Local);
         timeoff.ShowOffset(new DateTime(2007, 11, 4, 1, 0, 0),
TimeZoneInfo.Local);
         timeoff.ShowOffset(new DateTime(2006, 12, 10, 15, 0, 0),
TimeZoneInfo.Local);
         timeoff.ShowOffset(new DateTime(2007, 3, 11, 2, 30, 0),
TimeZoneInfo.Local);
         timeoff.ShowOffset(DateTime.UtcNow, TimeZoneInfo.Local);
         timeoff.ShowOffset(new DateTime(2006, 6, 12, 11, 0, 0),
TimeZoneInfo.Utc);
         timeoff.ShowOffset(new DateTime(2007, 11, 4, 1, 0, 0),
TimeZoneInfo.Utc);
         timeoff.ShowOffset(new DateTime(2006, 12, 10, 3, 0, 0),
TimeZoneInfo.Utc);
         timeoff.ShowOffset(new DateTime(2007, 3, 11, 2, 30, 0),
TimeZoneInfo.Utc);
         timeoff.ShowOffset(DateTime.Now, TimeZoneInfo.Utc);
         timeoff.ShowOffset(new DateTime(2006, 6, 12, 11, 0, 0), cst);
         timeoff.ShowOffset(new DateTime(2007, 11, 4, 1, 0, 0), cst);
         timeoff.ShowOffset(new DateTime(2006, 12, 10, 15, 0, 0), cst);
         timeoff.ShowOffset(new DateTime(2007, 3, 11, 2, 30, 0, 0), cst);
         timeoff.ShowOffset(new DateTime(2007, 11, 14, 00, 00, 00,
DateTimeKind.Local), cst);
      }

      private void ShowOffset(DateTime time, TimeZoneInfo timeZone)
      {
```

```
         DateTime convertedTime = time;
         TimeSpan offset;

         if (time.Kind == DateTimeKind.Local && !
timeZone.Equals(TimeZoneInfo.Local))
            convertedTime = TimeZoneInfo.ConvertTime(time,
TimeZoneInfo.Local, timeZone);
         else if (time.Kind == DateTimeKind.Utc && !
timeZone.Equals(TimeZoneInfo.Utc))
            convertedTime = TimeZoneInfo.ConvertTime(time,
TimeZoneInfo.Utc, timeZone);

         offset = timeZone.GetUtcOffset(time);
         if (time == convertedTime)
         {
            Console.WriteLine("{0} {1} ", time,
                              timeZone.IsDaylightSavingTime(time) ?
timeZone.DaylightName : timeZone.StandardName);
            Console.WriteLine("   It differs from UTC by {0} hours, {1}
minutes.",
                               offset.Hours,
                               offset.Minutes);
         }
         else
         {
            Console.WriteLine("{0} {1} ", time,
                              time.Kind == DateTimeKind.Utc ? "UTC" :
TimeZoneInfo.Local.Id);
            Console.WriteLine("   converts to {0} {1}.",
                              convertedTime,
                              timeZone.Id);
            Console.WriteLine("   It differs from UTC by {0} hours, {1}
minutes.",
                              offset.Hours, offset.Minutes);
         }
         Console.WriteLine();
      }
   }
}
// The example produces the following output:
//
//       6/12/2006 11:00:00 AM Pacific Daylight Time
//          It differs from UTC by -7 hours, 0 minutes.
//
//       11/4/2007 1:00:00 AM Pacific Standard Time
//          It differs from UTC by -8 hours, 0 minutes.
//
//       12/10/2006 3:00:00 PM Pacific Standard Time
//          It differs from UTC by -8 hours, 0 minutes.
//
//       3/11/2007 2:30:00 AM Pacific Standard Time
//          It differs from UTC by -8 hours, 0 minutes.
//
//       2/2/2007 8:35:46 PM UTC
//          converts to 2/2/2007 12:35:46 PM Pacific Standard Time.
```

```
//          It differs from UTC by -8 hours, 0 minutes.
//
//       6/12/2006 11:00:00 AM UTC
//          It differs from UTC by 0 hours, 0 minutes.
//
//       11/4/2007 1:00:00 AM UTC
//          It differs from UTC by 0 hours, 0 minutes.
//
//       12/10/2006 3:00:00 AM UTC
//          It differs from UTC by 0 hours, 0 minutes.
//
//       3/11/2007 2:30:00 AM UTC
//          It differs from UTC by 0 hours, 0 minutes.
//
//       2/2/2007 12:35:46 PM Pacific Standard Time
//          converts to 2/2/2007 8:35:46 PM UTC.
//          It differs from UTC by 0 hours, 0 minutes.
//
//       6/12/2006 11:00:00 AM Central Daylight Time
//          It differs from UTC by -5 hours, 0 minutes.
//
//       11/4/2007 1:00:00 AM Central Standard Time
//          It differs from UTC by -6 hours, 0 minutes.
//
//       12/10/2006 3:00:00 PM Central Standard Time
//          It differs from UTC by -6 hours, 0 minutes.
//
//       3/11/2007 2:30:00 AM Central Standard Time
//          It differs from UTC by -6 hours, 0 minutes.
//
//       11/14/2007 12:00:00 AM Pacific Standard Time
//          converts to 11/14/2007 2:00:00 AM Central Standard Time.
//          It differs from UTC by -6 hours, 0 minutes.
```

Remarks

The returned time span includes any differences due to the application of adjustment rules to the current time zone. It differs from the BaseUtcOffset property, which returns the difference between Coordinated Universal Time (UTC) and the time zone's standard time and, therefore, does not take adjustment rules into account.

If the `dateTime` parameter's Kind property does not correspond to the time zone object, this method performs the necessary conversion before returning a result. For example, this can occur if the Kind property is DateTimeKind.Local but the time zone object is not the local time zone. If `dateTime` is ambiguous, or if the converted time is ambiguous, this method interprets the ambiguous time as a standard time. If `dateTime` is invalid, this method returns a TimeSpan object that reflects the difference between UTC and the time zone's standard time.

The TimeZoneInfo.GetUtcOffset(DateTime) method is similar in operation to the GetUtcOffset method of the TimeZone class.

See also

- BaseUtcOffset

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetUtcOffset(DateTimeOffset)

Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.

C#

```
public TimeSpan GetUtcOffset (DateTimeOffset dateTimeOffset);
```

Parameters

`dateTimeOffset` DateTimeOffset

The date and time to determine the offset for.

Returns

TimeSpan

An object that indicates the time difference between Coordinated Universal Time (UTC) and the current time zone.

Remarks

The returned time span includes any differences due to the application of adjustment rules to the current time zone. It differs from the BaseUtcOffset property, which returns the difference between Coordinated Universal Time (UTC) and the time zone's standard time and, therefore, does not take adjustment rules into account.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.HasSameRules(TimeZoneInfo) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the current object and another TimeZoneInfo object have the same adjustment rules.

```
C#

public bool HasSameRules (TimeZoneInfo other);
```

Parameters

other TimeZoneInfo

A second object to compare with the current TimeZoneInfo object.

Returns

Boolean

`true` if the two time zones have identical adjustment rules and an identical base offset; otherwise, `false`.

Exceptions

ArgumentNullException

The `other` parameter is `null`.

Examples

Typically, a number of time zones defined in the registry on Windows and the ICU Library on Linux and macOS have the same offset from Coordinated Universal Time (UTC) and the same adjustment rules. The following example displays a list of these time zones to the console.

C#

```
ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
TimeZoneInfo[] timeZoneArray = new TimeZoneInfo[timeZones.Count];
timeZones.CopyTo(timeZoneArray, 0);
// Iterate array from top to bottom
for (int ctr = timeZoneArray.GetUpperBound(0); ctr >= 1; ctr--)
{
   // Get next item from top
   TimeZoneInfo thisTimeZone = timeZoneArray[ctr];
   for (int compareCtr = 0; compareCtr <= ctr - 1; compareCtr++)
   {
      // Determine if time zones have the same rules
      if (thisTimeZone.HasSameRules(timeZoneArray[compareCtr]))
      {
         Console.WriteLine("{0} has the same rules as {1}",
                           thisTimeZone.StandardName,
                           timeZoneArray[compareCtr].StandardName);
      }
   }
}
```

Remarks

Like the TimeZoneInfo.Equals(TimeZoneInfo) method, the HasSameRules method indicates whether two time zones have the same base offset (as defined by the BaseUtcOffset property) and the same adjustment rules. Unlike the TimeZoneInfo.Equals(TimeZoneInfo) method, HasSameRules does not compare time zone identifiers (as defined by the Id property).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.IsAmbiguousTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.

Overloads

Expand table

IsAmbiguousTime(DateTime)	Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.
IsAmbiguousTime(DateTimeOffset)	Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.

IsAmbiguousTime(DateTime)

Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.

C#

```
public bool IsAmbiguousTime (DateTime dateTime);
```

Parameters

`dateTime` DateTime

A date and time value.

Returns

Boolean

`true` if the `dateTime` parameter is ambiguous; otherwise, `false`.

Exceptions

ArgumentException

The Kind property of the `dateTime` value is Local and `dateTime` is an invalid time.

Examples

In the Pacific Time zone, daylight saving time ends at 2:00 A.M. on November 4, 2007. The following example passes the time at one-minute intervals from 12:59 A.M. on November 4, 2007, to 2:01 A.M. on November 4, 2007, to the IsAmbiguousTime(DateTime) method of a TimeZoneInfo object that represents the Pacific Time zone. The console output indicates that all times from 1:00 A.M. on November 4, 2007, to 1:59 A.M. on November 4, 2007, are ambiguous.

C#

```
   // Specify DateTimeKind in Date constructor
   DateTime baseTime = new DateTime(2007, 11, 4, 0, 59, 00,
DateTimeKind.Unspecified);
   DateTime newTime;

   // Get Pacific Standard Time zone
   TimeZoneInfo pstZone = TimeZoneInfo.FindSystemTimeZoneById("Pacific
Standard Time");

   // List possible ambiguous times for 63-minute interval, from 12:59 AM
to 2:01 AM
   for (int ctr = 0; ctr < 63; ctr++)
   {
      // Because of assignment, newTime.Kind is also
DateTimeKind.Unspecified
      newTime = baseTime.AddMinutes(ctr);
      Console.WriteLine("{0} is ambiguous: {1}", newTime,
pstZone.IsAmbiguousTime(newTime));
   }
```

Remarks

An ambiguous time falls within a range of times for the current time zone. This means it can be either a standard time or a time that results from the application of

an adjustment rule. Typically, ambiguous times result when the clock is set to return to standard time from daylight saving time. See the Example section for an illustration.

Coordinated Universal Time (UTC) has no ambiguous times; neither do time zones that do not support daylight saving time. Therefore, these time zones have no adjustment rules and calls to the IsAmbiguousTime method always return `false`.

For time zones that do observe daylight saving time, the precise behavior of this method depends on the relationship between the Kind property and the TimeZoneInfo object, as the following table shows.

Expand table

TimeZoneInfo object type	Kind property value	Behavior
TimeZoneInfo.Local	DateTimeKind.Local or DateTimeKind.Unspecified	Determines whether the `dateTime` parameter is ambiguous.
TimeZoneInfo.Local	DateTimeKind.Utc	Converts `dateTime` to the local time and then determines whether that time is ambiguous.
TimeZoneInfo.Utc	DateTimeKind.Utc or DateTimeKind.Unspecified	Returns `false`.
TimeZoneInfo.Utc	DateTimeKind.Local	If `dateTime` is ambiguous, assumes it is a standard time, converts it to UTC, and returns `false`.
Any other time zone.	DateTimeKind.Local or DateTimeKind.Utc	Converts `dateTime` to the time in the specified time zone and then determines whether that time is ambiguous.
Any other time zone.	DateTimeKind.Unspecified	Determines whether `dateTime` is ambiguous.

See also

- IsInvalidTime(DateTime)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsAmbiguousTime(DateTimeOffset)

Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.

```
C#

public bool IsAmbiguousTime (DateTimeOffset dateTimeOffset);
```

Parameters

`dateTimeOffset` DateTimeOffset

A date and time.

Returns

Boolean

`true` if the `dateTimeOffset` parameter is ambiguous in the current time zone; otherwise, `false`.

Remarks

An ambiguous time falls within a range of times for the current time zone. This means that it can be either a standard time or a time that results from the application of an adjustment rule. Typically, ambiguous times result when the clock is set to return to standard time from daylight saving time. Coordinated Universal Time (UTC) has no ambiguous times; neither do time zones that do not support daylight saving time.

The precise behavior of the method depends on the relationship between the Offset property of the `dateTimeOffset` parameter and the current time zone. If the value of the Offset property is a possible offset from the UTC of the current time zone, the method determines whether that date and time is ambiguous. Otherwise, it converts `dateTimeOffset` to the time in the current time zone, and then determines whether that date and time is ambiguous.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.IsDaylightSavingTime Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a specified date and time falls in the range of daylight saving time for the current TimeZoneInfo object's time zone.

Overloads

Expand table

IsDaylightSavingTime(DateTime)	Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.
IsDaylightSavingTime(DateTimeOffset)	Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.

IsDaylightSavingTime(DateTime)

Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.

C#

```
public bool IsDaylightSavingTime (DateTime dateTime);
```

Parameters

dateTime DateTime

A date and time value.

Returns

Boolean

`true` if the `dateTime` parameter is a daylight saving time; otherwise, `false`.

Exceptions

ArgumentException

The Kind property of the `dateTime` value is Local and `dateTime` is an invalid time.

Examples

The following example defines a method named `DisplayDateWithTimeZoneName` that uses the TimeZoneInfo.IsDaylightSavingTime method to determine whether to display a time zone's standard time name or daylight saving time name.

C#

```
private void DisplayDateWithTimeZoneName(DateTime date1, TimeZoneInfo
timeZone)
{
   Console.WriteLine("The time is {0:t} on {0:d} {1}",
                     date1,
                     timeZone.IsDaylightSavingTime(date1) ?
                         timeZone.DaylightName : timeZone.StandardName);
}
// The example displays output similar to the following:
//    The time is 1:00 AM on 4/2/2006 Pacific Standard Time
```

Remarks

The return value of TimeZoneInfo.IsDaylightSavingTime is affected by the relationship between the time zone represented by the TimeZoneInfo object and the Kind property of the `dateTime` parameter, as the following table shows.

Expand table

TimeZoneInfo object	DateTime.Kind property	Result
TimeZoneInfo.Local	`DateTimeKind.Local`	Determines whether `dateTime` is daylight saving time.
TimeZoneInfo.Local	`DateTimeKind.Utc`	Converts `dateTime` from Coordinated Universal Time (UTC)

TimeZoneInfo object	DateTime.Kind property	Result
		to local time and determines whether it is daylight saving time.
TimeZoneInfo.Local	`DateTimeKind.Unspecified`	Assumes that `dateTime` represents local time and determines whether it is daylight saving time.
TimeZoneInfo.Utc	`DateTimeKind.Local`, `DateTimeKind.Unspecified`, or `DateTimeKind.Utc`	Returns `false` (UTC does not support daylight saving time).
Any other TimeZoneInfo object.	`DateTimeKind.Local`	Converts the local time to the equivalent time of the TimeZoneInfo object and then determines whether the latter is daylight saving time.
Any other TimeZoneInfo object.	`DateTimeKind.Utc`	Converts UTC to the equivalent time of the TimeZoneInfo object and then determines whether the latter is daylight saving time.
Any other TimeZoneInfo object.	`DateTimeKind.Unspecified`	Determines whether `dateTime` is daylight saving time.

If the time zone represented by the TimeZoneInfo object does not support daylight saving time, the method always returns `false`. A number of time zones, including Utc, do not observe daylight saving time. To determine whether a time zone supports daylight saving time, retrieve the value of its SupportsDaylightSavingTime property.

If the `dateTime` parameter specifies an ambiguous time in the current object's time zone, the TimeZoneInfo.IsDaylightSavingTime method interprets `dateTime` as standard time and returns `false` if its Kind property is DateTimeKind.Local or DateTimeKind.Unspecified. If the Kind property is DateTimeKind.Utc, this method will select the correct ambiguous time and indicate whether it is a daylight saving time.

Because the TimeZoneInfo.IsDaylightSavingTime(DateTime) method can return `false` for a date and time that is ambiguous (that is, a date and time that can represent either a standard time or a daylight saving time in a particular time zone), the TimeZoneInfo.IsAmbiguousTime(DateTime) method can be paired with the IsDaylightSavingTime(DateTime) method to determine whether a time may be a daylight saving time. Because an ambiguous time is one that can be both a daylight

saving time and a standard time, the IsAmbiguousTime(DateTime) method can be called first to determine whether a date and time may be a daylight saving time. If the method returns `false`, the IsDaylightSavingTime(DateTime) method can be called to determine whether the DateTime value is a daylight saving time. The following example illustrates this technique.

C#

```
DateTime unclearDate = new DateTime(2007, 11, 4, 1, 30, 0);
// Test if time is ambiguous.
Console.WriteLine("In the {0}, {1} is {2}ambiguous.",
                  TimeZoneInfo.Local.DisplayName,
                  unclearDate,
                  TimeZoneInfo.Local.IsAmbiguousTime(unclearDate) ? "" :
"not ");
// Test if time is DST.
Console.WriteLine("In the {0}, {1} is {2}daylight saving time.",
                  TimeZoneInfo.Local.DisplayName,
                  unclearDate,
                  TimeZoneInfo.Local.IsDaylightSavingTime(unclearDate) ?
"" : "not ");
Console.WriteLine();
// Report time as DST if it is either ambiguous or DST.
if (TimeZoneInfo.Local.IsAmbiguousTime(unclearDate) ||
    TimeZoneInfo.Local.IsDaylightSavingTime(unclearDate))
    Console.WriteLine("{0} may be daylight saving time in {1}.",
                      unclearDate, TimeZoneInfo.Local.DisplayName);
// The example displays the following output:
//    In the (GMT-08:00) Pacific Time (US & Canada), 11/4/2007 1:30:00 AM
is ambiguous.
//    In the (GMT-08:00) Pacific Time (US & Canada), 11/4/2007 1:30:00 AM
is not daylight saving time.
//
//    11/4/2007 1:30:00 AM may be daylight saving time in (GMT-08:00)
Pacific Time (US & Canada).
```

If the `dateTime` parameter specifies an invalid time, the method call throws an ArgumentException if the value of the `dateTime` parameter's Kind property is DateTimeKind.Local; otherwise, the method returns `false`.

Call the TimeZoneInfo.IsDaylightSavingTime method to determine whether to use a time zone's StandardName value or its DaylightName value when displaying the time zone name. See the Example section for an illustration.

See also

- SupportsDaylightSavingTime

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

IsDaylightSavingTime(DateTimeOffset)

Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.

C#

```
public bool IsDaylightSavingTime (DateTimeOffset dateTimeOffset);
```

Parameters

`dateTimeOffset` DateTimeOffset

A date and time value.

Returns

Boolean

`true` if the `dateTimeOffset` parameter is a daylight saving time; otherwise, `false`.

Remarks

The return value of TimeZoneInfo.IsDaylightSavingTime is affected by the relationship between the time zone represented by the TimeZoneInfo object and the Offset property of the `dateTimeOffset` parameter. If `dateTimeOffset` does not correspond to the current time zone's offset from Coordinated Universal Time (UTC),

the method converts that time to the time in the current time zone. It then determines whether that date and time is a daylight saving time.

If the time zone represented by the TimeZoneInfo object does not support daylight saving time, the method always returns `false`. To determine whether a time zone supports daylight saving time, retrieve the value of its SupportsDaylightSavingTime property.

See also

- SupportsDaylightSavingTime

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.IsInvalidTime(DateTime) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a particular date and time is invalid.

```
C#

public bool IsInvalidTime (DateTime dateTime);
```

Parameters

dateTime DateTime

A date and time value.

Returns

Boolean

`true` if `dateTime` is invalid; otherwise, `false`.

Examples

In the Pacific Time zone, daylight saving time begins at 2:00 A.M. on April 2, 2006. The following code passes the time at one-minute intervals from 1:59 A.M. on April 2, 2006, to 3:01 A.M. on April 2, 2006, to the IsInvalidTime method of a TimeZoneInfo object that represents the Pacific Time zone. The console output indicates that all times from 2:00 A.M. on April 2, 2006, to 2:59 A.M. on April 2, 2006, are invalid.

```
C#

// Specify DateTimeKind in Date constructor
DateTime baseTime = new DateTime(2007, 3, 11, 1, 59, 0,
DateTimeKind.Unspecified);
DateTime newTime;
```

```
// Get Pacific Standard Time zone
TimeZoneInfo pstZone = TimeZoneInfo.FindSystemTimeZoneById("Pacific Standard
Time");

// List possible invalid times for a 63-minute interval, from 1:59 AM to
3:01 AM
for (int ctr = 0; ctr < 63; ctr++)
{
   // Because of assignment, newTime.Kind is also DateTimeKind.Unspecified
   newTime = baseTime.AddMinutes(ctr);
   Console.WriteLine("{0} is invalid: {1}", newTime,
pstZone.IsInvalidTime(newTime));
}
```

Remarks

An invalid time falls within a range of times for the current time zone that cannot be mapped to Coordinated Universal Time (UTC) due to the application of an adjustment rule. Typically, invalid times occur when the time moves ahead for daylight saving time. See the Example section for an illustration.

The value of the Kind property of the `dateTime` parameter affects whether `dateTime` represents an invalid time, as the following table shows.

Expand table

DateTime.Kind property	TimeZoneInfo object (if applicable)	Behavior
DateTimeKind.Local	TimeZoneInfo.Local	Determines whether the time is invalid.
DateTimeKind.Local	TimeZoneInfo.Utc or a non-local time zone.	Converts `dateTime` to the time of the TimeZoneInfo object and returns `false`.
DateTimeKind.Unspecified	Not applicable.	Assumes `dateTime` is the time of the TimeZoneInfo object and determines whether it is invalid.
DateTimeKind.Utc	Not applicable.	Returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsAmbiguousTime

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ToSerializedString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the current TimeZoneInfo object to a serialized string.

C#

```
public string ToSerializedString ();
```

Returns

String

A string that represents the current TimeZoneInfo object.

Examples

The following example tries to retrieve the Antarctica/South Pole time zone from the local system. If it fails, the code tries to retrieve information about the time zone from a text file in the application directory. If this attempt fails, the code creates the time zone and writes information about it to the text file.

C#

```
private TimeZoneInfo InitializeTimeZone()
{
   TimeZoneInfo southPole = null;
   // Determine if South Pole time zone is defined in system
   try
   {
      southPole = TimeZoneInfo.FindSystemTimeZoneById("Antarctica/South Pole
Standard Time");
   }
   // Time zone does not exist; create it, store it in a text file, and
return it
   catch
```

```
    {
        const string filename = @".\TimeZoneInfo.txt";
        bool found = false;

        if (File.Exists(filename))
        {
            StreamReader reader = new StreamReader(filename);
            string timeZoneInfo;
            while (reader.Peek() >= 0)
            {
                timeZoneInfo = reader.ReadLine();
                if (timeZoneInfo.Contains("Antarctica/South Pole"))
                {
                    southPole = TimeZoneInfo.FromSerializedString(timeZoneInfo);
                    reader.Close();
                    found = true;
                    break;
                }
            }
        }
        if (! found)
        {
            // Define transition times to/from DST
            TimeZoneInfo.TransitionTime startTransition =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1, 2,
0, 0), 10, 1, DayOfWeek.Sunday);
            TimeZoneInfo.TransitionTime endTransition =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1, 2,
0, 0), 3, 3, DayOfWeek.Sunday);
            // Define adjustment rule
            TimeSpan delta = new TimeSpan(1, 0, 0);
            TimeZoneInfo.AdjustmentRule adjustment =
TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new DateTime(1989, 10, 1),
DateTime.MaxValue.Date, delta, startTransition, endTransition);
            // Create array for adjustment rules
            TimeZoneInfo.AdjustmentRule[] adjustments = {adjustment};
            // Define other custom time zone arguments
            string displayName = "(GMT+12:00) Antarctica/South Pole";
            string standardName = "Antarctica/South Pole Standard Time";
            string daylightName = "Antarctica/South Pole Daylight Time";
            TimeSpan offset = new TimeSpan(12, 0, 0);
            southPole = TimeZoneInfo.CreateCustomTimeZone(standardName, offset,
displayName, standardName, daylightName, adjustments);
            // Write time zone to the file
            StreamWriter writer = new StreamWriter(filename, true);
            writer.WriteLine(southPole.ToSerializedString());
            writer.Close();
        }
    }
    return southPole;
}
```

Remarks

Applications that rely on time zones that are not typically defined in the registry of Windows systems or the ICU Library on Linux and macOScan can use the CreateCustomTimeZone method to instantiate the necessary time zones as TimeZoneInfo objects. The application can then call the ToSerializedString method to convert the time zone object to a string.

The TimeZoneInfo object should also be stored in a location where the application can retrieve it when needed. Possible locations include:

- The registry on Windows systems.
- An application resource file.
- An external file, such as a text file.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FromSerializedString(String)
- How to: Save Time Zones to an Embedded Resource

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the current TimeZoneInfo object's display name.

C#

```
public override string ToString ();
```

Returns

String

The value of the DisplayName property of the current TimeZoneInfo object.

Remarks

When assigning a collection that contains TimeZoneInfo objects to a list control such as the ListBox control, the control automatically calls the ToString method to extract a string that describes each object to be represented in the list. As a result, you do not need to store a meaningful description of the object as a string, and the user does not need to use that string to extract the object from the collection. See How to: Enumerate Time Zones Present on a Computer for more detail.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TryConvertIanaIdToWindowsId(String, String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an IANA time zone ID to a Windows ID.

```
C#

public static bool TryConvertIanaIdToWindowsId (string ianaId, out string?
windowsId);
```

Parameters

`ianaId` String

The IANA time zone ID.

`windowsId` String

When this method returns, contains the Windows ID that corresponds to the specified IANA ID.

Returns

Boolean

`true` if the ID conversion succeeded, `false` otherwise.

Remarks

This API is exclusively supported on .NET 6 and later versions, and only when the application utilizes the ICU library. However, in .NET, the ICU library is not always used by default, particularly in the following scenarios:

- When the application runs on an operating system that does not have the ICU library, and the app does not utilize the app-local ICU feature.

- When the globalization invariant mode or NLS mode is enabled on Windows.

If the application is running on a Windows OS version that lacks the ICU library, it has the option to opt-in and enable the usage of the ICU library through the app-local ICU feature.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.TryConvertWindowsIdToIanaId Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryConvertWindowsIdToIanaId(String, String)	Tries to convert a Windows time zone ID to an IANA ID.
TryConvertWindowsIdToIanaId(String, String, String)	Tries to convert a Windows time zone ID to an IANA ID.

TryConvertWindowsIdToIanaId(String, String)

Tries to convert a Windows time zone ID to an IANA ID.

C#

```
public static bool TryConvertWindowsIdToIanaId (string windowsId, out
string? ianaId);
```

Parameters

`windowsId` String

The Windows time zone ID.

`ianaId` String

When this method returns, contains the IANA ID that corresponds to the specified Windows ID.

Returns

Boolean

`true` if the ID conversion succeeded, `false` otherwise.

Remarks

This API is exclusively supported on .NET 6.0 and later versions. It is also supported when the application utilizes the ICU library, otherwise the API will fail. However, in .NET, the ICU library is not always used by default, particularly in the following scenarios:

- When the application runs on an operating system that does not have the ICU library, and the app does not utilize the APP-Local ICU feature.
- When the Globalization Invariant Mode or NLS mode is enabled on Windows OS.

If the application is running on a Windows OS version that lacks the ICU library, it has the option to opt-in and enable the usage of the ICU library through the App-Local ICU feature.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryConvertWindowsIdToIanaId(String, String, String)

Tries to convert a Windows time zone ID to an IANA ID.

C#

```
public static bool TryConvertWindowsIdToIanaId (string windowsId, string?
region, out string? ianaId);
```

Parameters

`windowsId` String

The Windows time zone ID.

`region` String

The ISO 3166 code for the country/region.

`ianaId` String

When this method returns, contains the IANA ID that corresponds to the specified Windows ID.

Returns

Boolean

`true` if the ID conversion succeeded, `false` otherwise.

Remarks

This API is exclusively supported on .NET 6.0 and later versions. It is also supported when the application utilizes the ICU library, otherwise the API will fail. However, in .NET, the ICU library is not always used by default, particularly in the following scenarios:

- When the application runs on an operating system that does not have the ICU library, and the app does not utilize the App-local ICU feature.
- When the Globalization Invariant Mode or NLS mode is enabled on Windows OS.

If the application is running on a Windows OS version that lacks the ICU library, it has the option to opt-in and enable the usage of the ICU library through the App-Local ICU feature.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TryFindSystemTimeZoneById(String, TimeZoneInfo) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves a TimeZoneInfo object by time zone name.

C#

```
public static bool TryFindSystemTimeZoneById (string id, out TimeZoneInfo?
timeZoneInfo);
```

Parameters

id String

The time zone name.

timeZoneInfo TimeZoneInfo

When this method returns, contains the retrieved TimeZoneInfo or `null` if no valid time zone was found.

Returns

Boolean

`true` if the TimeZoneInfo object was successfully retrieved; otherwise, `false`.

Remarks

This function wraps the logic necessary to keep the private SystemTimeZones cache in working order.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.IDeserializationCallback.OnDeserialization(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Runs when the deserialization of an object has been completed.

C#

```
void IDeserializationCallback.OnDeserialization (object sender);
```

Parameters

sender Object

The object that initiated the callback. The functionality for this parameter is not currently implemented.

Implements

OnDeserialization(Object)

Exceptions

SerializationException

The TimeZoneInfo object contains invalid or corrupted data.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data needed to serialize the current TimeZoneInfo object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo object to populate with data.

`context` StreamingContext

The destination for this serialization (see StreamingContext).

Implements

GetObjectData(SerializationInfo, StreamingContext)

Exceptions

ArgumentNullException

The `info` parameter is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides information about a time zone adjustment, such as the transition to and from daylight saving time.

C#

```
public sealed class TimeZoneInfo.AdjustmentRule :
IEquatable<TimeZoneInfo.AdjustmentRule>,
System.Runtime.Serialization.IDeserializationCallback,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → TimeZoneInfo.AdjustmentRule

Implements IEquatable<TimeZoneInfo.AdjustmentRule> , IDeserializationCallback , ISerializable

Examples

The following example retrieves all time zones defined on the local system and displays complete information about their adjustment rules.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
   Last = 5,
}

private static void ShowStartAndEndDates()
{
   // Get all time zones from system
   ReadOnlyCollection<TimeZoneInfo> timeZones =
```

```
TimeZoneInfo.GetSystemTimeZones();
   string[] monthNames =
CultureInfo.CurrentCulture.DateTimeFormat.MonthNames;
   // Get each time zone
   foreach (TimeZoneInfo timeZone in timeZones)
   {
      TimeZoneInfo.AdjustmentRule[] adjustments =
timeZone.GetAdjustmentRules();
      // Display message for time zones with no adjustments
      if (adjustments.Length == 0)
      {
         Console.WriteLine("{0} has no adjustment rules",
timeZone.StandardName);
      }
      else
      {
         // Handle time zones with 1 or 2+ adjustments differently
         bool showCount = false;
         int ctr = 0;
         string spacer = "";

         Console.WriteLine("{0} Adjustment rules", timeZone.StandardName);
         if (adjustments.Length > 1)
         {
            showCount = true;
            spacer = "   ";
         }
         // Iterate adjustment rules
         foreach (TimeZoneInfo.AdjustmentRule adjustment in adjustments)
         {
            if (showCount)
            {
               Console.WriteLine("   Adjustment rule #{0}", ctr+1);
               ctr++;
            }
            // Display general adjustment information
            Console.WriteLine("{0}   Start Date: {1:D}", spacer,
adjustment.DateStart);
            Console.WriteLine("{0}   End Date: {1:D}", spacer,
adjustment.DateEnd);
            Console.WriteLine("{0}   Time Change: {1}:{2:00} hours", spacer,
                              adjustment.DaylightDelta.Hours,
adjustment.DaylightDelta.Minutes);
            // Get transition start information
            TimeZoneInfo.TransitionTime transitionStart =
adjustment.DaylightTransitionStart;
            Console.Write("{0}   Annual Start: ", spacer);
            if (transitionStart.IsFixedDateRule)
            {
               Console.WriteLine("On {0} {1} at {2:t}",
                                 monthNames[transitionStart.Month - 1],
                                 transitionStart.Day,
                                 transitionStart.TimeOfDay);
            }
            else
```

```
            {
               Console.WriteLine("The {0} {1} of {2} at {3:t}",

((WeekOfMonth)transitionStart.Week).ToString(),
                                 transitionStart.DayOfWeek.ToString(),
                                 monthNames[transitionStart.Month - 1],
                                 transitionStart.TimeOfDay);
            }
            // Get transition end information
            TimeZoneInfo.TransitionTime transitionEnd =
adjustment.DaylightTransitionEnd;
            Console.Write("{0}   Annual End: ", spacer);
            if (transitionEnd.IsFixedDateRule)
            {
               Console.WriteLine("On {0} {1} at {2:t}",
                                 monthNames[transitionEnd.Month - 1],
                                 transitionEnd.Day,
                                 transitionEnd.TimeOfDay);
            }
            else
            {
               Console.WriteLine("The {0} {1} of {2} at {3:t}",

((WeekOfMonth)transitionEnd.Week).ToString(),
                                 transitionEnd.DayOfWeek.ToString(),
                                 monthNames[transitionEnd.Month - 1],
                                 transitionEnd.TimeOfDay);
            }
         }
      }
      Console.WriteLine();
   }
}
```

The following is a small portion of the output that is generated by the example. The exact output will vary depending on the operating system and the date on which the example is run.

```
Morocco Standard Time Adjustment rules
   Adjustment rule #1
      Start Date: Tuesday, January 01, 2008
      End Date: Wednesday, December 31, 2008
      Time Change: 1:00 hours
      Annual Start: The Last Saturday of May at 11:59 PM
      Annual End: The Last Sunday of August at 11:59 PM
   Adjustment rule #2
      Start Date: Thursday, January 01, 2009
      End Date: Thursday, December 31, 2009
      Time Change: 1:00 hours
      Annual Start: The Last Sunday of May at 11:59 PM
```

```
      Annual End: The Third Thursday of August at 11:59 PM

Coordinated Universal Time has no adjustment rules

GMT Standard Time Adjustment rules
   Start Date: Monday, January 01, 0001
   End Date: Friday, December 31, 9999
   Time Change: 1:00 hours
   Annual Start: The Last Sunday of March at 1:00 AM
   Annual End: The Last Sunday of October at 2:00 AM

Greenwich Standard Time has no adjustment rules

W. Europe Standard Time Adjustment rules
   Start Date: Monday, January 01, 0001
   End Date: Friday, December 31, 9999
   Time Change: 1:00 hours
   Annual Start: The Last Sunday of March at 2:00 AM
   Annual End: The Last Sunday of October at 3:00 AM

Central Europe Standard Time Adjustment rules
   Start Date: Monday, January 01, 0001
   End Date: Friday, December 31, 9999
   Time Change: 1:00 hours
   Annual Start: The Last Sunday of March at 2:00 AM
   Annual End: The Last Sunday of October at 3:00 AM

Romance Standard Time Adjustment rules
   Start Date: Monday, January 01, 0001
   End Date: Friday, December 31, 9999
   Time Change: 1:00 hours
   Annual Start: The Last Sunday of March at 2:00 AM
   Annual End: The Last Sunday of October at 3:00 AM

Central European Standard Time Adjustment rules
   Start Date: Monday, January 01, 0001
   End Date: Friday, December 31, 9999
   Time Change: 1:00 hours
   Annual Start: The Last Sunday of March at 2:00 AM
   Annual End: The Last Sunday of October at 3:00 AM

W. Central Africa Standard Time has no adjustment rules
```

Remarks

The TimeZoneInfo.AdjustmentRule class defines the effective start and end dates of a particular time change to and from daylight saving time, respectively, as well as its delta (the exact amount by which the adjustment causes the time zone's standard time to change). In addition, two properties return TimeZoneInfo.TransitionTime objects that define when each transition to and from standard time occurs.

> **Note**
>
> An instance of the **TimeZoneInfo.AdjustmentRule** class is immutable. Once an object has been created, its values cannot be modified.

To create a TimeZoneInfo.AdjustmentRule object, call the `static` (`Shared` in Visual Basic) TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule method. You can then supply an array of TimeZoneInfo.AdjustmentRule objects to two of the overloads of the TimeZoneInfo.CreateCustomTimeZone method. To retrieve the adjustment rules of a particular time zone, call its TimeZoneInfo.GetAdjustmentRules method, which returns an array of TimeZoneInfo.AdjustmentRule objects.

Properties

Expand table

BaseUtcOffsetDelta	Gets the time difference with the base UTC offset for the time zone during the adjustment-rule period.
DateEnd	Gets the date when the adjustment rule ceases to be in effect.
DateStart	Gets the date when the adjustment rule takes effect.
DaylightDelta	Gets the amount of time that is required to form the time zone's daylight saving time. This amount of time is added to the time zone's offset from Coordinated Universal Time (UTC).
DaylightTransitionEnd	Gets information about the annual transition from daylight saving time back to standard time.
DaylightTransitionStart	Gets information about the annual transition from standard time to daylight saving time.

Methods

Expand table

CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime)	Creates a new adjustment rule for a particular time zone.
CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZone	Creates a new adjustment rule for a particular time zone.

Info+TransitionTime, TimeSpan)	
Equals(Object)	Indicates whether the current instance is equal to another instance.
Equals(TimeZoneInfo+AdjustmentRule)	Determines whether the current TimeZoneInfo.AdjustmentRule object is equal to a second TimeZoneInfo.AdjustmentRule object.
GetHashCode()	Serves as a hash function for hashing algorithms and data structures such as hash tables.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IDeserializationCallback.OnDeserialization(Object)	Runs when the deserialization of a TimeZoneInfo.AdjustmentRule object is completed.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data that is required to serialize this object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TimeZoneInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.AdjustmentRule.BaseUtcOffsetDelta Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the time difference with the base UTC offset for the time zone during the adjustment-rule period.

C#

```
public TimeSpan BaseUtcOffsetDelta { get; }
```

Property Value

TimeSpan

A TimeSpan instance representing the time difference.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.AdjustmentRule.DateEnd Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the date when the adjustment rule ceases to be in effect.

C#

```
public DateTime DateEnd { get; }
```

Property Value

DateTime

A DateTime value that indicates the end date of the adjustment rule.

Examples

The following example displays information about all of the time zones defined in the local computer's system registry, including the starting and ending dates of their adjustment rules.

C#

```
using System;
using System.Collections.ObjectModel;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
      DateTimeFormatInfo dateInfo =
CultureInfo.CurrentCulture.DateTimeFormat;

      foreach (var zone in timeZones)
```

```
      {
         Console.WriteLine("{0} transition time information:",
zone.StandardName);
         Console.WriteLine("   Time zone information: ");
         Console.WriteLine("      Base UTC Offset: {0}",
zone.BaseUtcOffset);
         Console.WriteLine("      Supports DST: {0}",
zone.SupportsDaylightSavingTime);

         TimeZoneInfo.AdjustmentRule[] adjustmentRules=
zone.GetAdjustmentRules();

         // Indicate that time zone has no adjustment rules
         if (adjustmentRules.Length == 0) {
            Console.WriteLine("      No adjustment rules defined.");
         }
         else {
            Console.WriteLine("      Adjustment Rules: {0}",
adjustmentRules.Length);
            // Iterate adjustment rules
            foreach (var adjustmentRule in adjustmentRules) {
               Console.WriteLine("   Adjustment rule from {0:d} to {1:d}:",
                                 adjustmentRule.DateStart,
                                 adjustmentRule.DateEnd);
               Console.WriteLine("      Delta: {0}",
adjustmentRule.DaylightDelta);
               // Get start of transition
               TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
               // Display information on floating date rule
               if (! daylightStart.IsFixedDateRule)
                  Console.WriteLine("      Begins at {0:t} on the {1} {2} of
{3}",
                                 daylightStart.TimeOfDay,
                                 (WeekOfMonth) daylightStart.Week,
                                 daylightStart.DayOfWeek,

dateInfo.GetMonthName(daylightStart.Month));
               // Display information on fixed date rule
               else
                  Console.WriteLine("      Begins at {0:t} on {1} {2}",
                                    daylightStart.TimeOfDay,

dateInfo.GetMonthName(daylightStart.Month),
                                    daylightStart.Day);

               // Get end of transition.
              TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
               // Display information on floating date rule.
               if (!daylightEnd.IsFixedDateRule)
                  Console.WriteLine("      Ends at {0:t} on the {1} {2} of
{3}",
                                 daylightEnd.TimeOfDay,
                                 (WeekOfMonth) daylightEnd.Week,
```

```
                                 daylightEnd.DayOfWeek,
                                 dateInfo.GetMonthName(daylightEnd.Month));
               // Display information on fixed date rule.
               else
                  Console.WriteLine("      Ends at {0:t} on {1} {2}",
                                    daylightEnd.TimeOfDay,

dateInfo.GetMonthName(daylightEnd.Month),
                                    daylightEnd.Day);
            }
         }
      }
   }

   private enum WeekOfMonth
   {
      First = 1,
      Second = 2,
      Third = 3,
      Fourth = 4,
      Last = 5,
   }
}
// A portion of the output from the example might appear as follows:
//       Tonga Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 13:00:00
//             Supports DST: False
//             No adjustment rules defined.
//       Samoa Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 13:00:00
//             Supports DST: True
//             Adjustment Rules: 4
//          Adjustment rule from 1/1/0001 to 12/31/2009:
//             Delta: 00:00:00
//             Begins at 12:00 AM on January 1
//             Ends at 12:00 AM on January 1
//          Adjustment rule from 1/1/2010 to 12/31/2010:
//             Delta: 01:00:00
//             Begins at 11:59 PM on the Last Saturday of September
//             Ends at 12:00 AM on the First Friday of January
//          Adjustment rule from 1/1/2011 to 12/31/2011:
//             Delta: 01:00:00
//             Begins at 3:00 AM on the Fourth Saturday of September
//             Ends at 4:00 AM on the First Saturday of April
//          Adjustment rule from 1/1/2012 to 12/31/9999:
//             Delta: 01:00:00
//             Begins at 12:00 AM on the Last Sunday of September
//             Ends at 1:00 AM on the First Sunday of April
//       Line Islands Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 14:00:00
//             Supports DST: False
//             No adjustment rules defined.
```

Remarks

The value of the DateEnd property is a date value without a time component.

Because the end date of the current adjustment rule is typically not known, you can assign `DateTime.MaxValue.Date` to the DateEnd property when you create a custom adjustment rule.

> **Important**
>
> Unless there is a compelling reason to do otherwise, you should define the adjustment rule's end date to occur within the time interval during which the time zone observes standard time. Unless there is a compelling reason to do so, you should not define the adjustment rule's end date to occur within the time interval during which the time zone observes daylight saving time. For example, if a time zone's transition from daylight saving time occurs on the third Sunday of March and its transition to daylight saving time occurs on the first Sunday of October, the effective end date of the adjustment rule should not be December 31 of a particular year, since that date occurs within the period of daylight saving time.

By default, the registry in Windows XP defines a single adjustment rule whose end date is Friday, December 31, 9999 (the value of `DateTime.MaxValue.Date`), for each time zone. For time zones in the United States, the registry in Windows Vista defines two adjustment rules:

- Monday, January 01, 0001, to Sunday, December 31, 2006.
- Monday, January 01, 2007, to Friday, December 31, 9999.

This means that, although time zone adjustment rules stored in the registry are useful for performing current time zone-related operations, they cannot be reliably used for retrieving historical time zone information. For information about defining a custom time zone with multiple adjustment rules that can be used in a historical time zone-aware application, see How to: Create Time Zones with Adjustment Rules.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule.DateStart Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the date when the adjustment rule takes effect.

C#

```
public DateTime DateStart { get; }
```

Property Value

DateTime

A DateTime value that indicates when the adjustment rule takes effect.

Examples

The following example displays information about all of the time zones defined in the local computer's system registry, including the starting and ending dates of their adjustment rules.

C#

```
using System;
using System.Collections.ObjectModel;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
      DateTimeFormatInfo dateInfo =
CultureInfo.CurrentCulture.DateTimeFormat;

      foreach (var zone in timeZones)
```

```
      {
         Console.WriteLine("{0} transition time information:",
zone.StandardName);
         Console.WriteLine("   Time zone information: ");
         Console.WriteLine("      Base UTC Offset: {0}",
zone.BaseUtcOffset);
         Console.WriteLine("      Supports DST: {0}",
zone.SupportsDaylightSavingTime);

         TimeZoneInfo.AdjustmentRule[] adjustmentRules=
zone.GetAdjustmentRules();

         // Indicate that time zone has no adjustment rules
         if (adjustmentRules.Length == 0) {
            Console.WriteLine("      No adjustment rules defined.");
         }
         else {
            Console.WriteLine("      Adjustment Rules: {0}",
adjustmentRules.Length);
            // Iterate adjustment rules
            foreach (var adjustmentRule in adjustmentRules) {
               Console.WriteLine("   Adjustment rule from {0:d} to {1:d}:",
                                 adjustmentRule.DateStart,
                                 adjustmentRule.DateEnd);
               Console.WriteLine("      Delta: {0}",
adjustmentRule.DaylightDelta);
               // Get start of transition
               TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
               // Display information on floating date rule
               if (! daylightStart.IsFixedDateRule)
                  Console.WriteLine("      Begins at {0:t} on the {1} {2} of
{3}",
                                 daylightStart.TimeOfDay,
                                 (WeekOfMonth) daylightStart.Week,
                                 daylightStart.DayOfWeek,

dateInfo.GetMonthName(daylightStart.Month));
               // Display information on fixed date rule
               else
                  Console.WriteLine("      Begins at {0:t} on {1} {2}",
                                    daylightStart.TimeOfDay,

dateInfo.GetMonthName(daylightStart.Month),
                                    daylightStart.Day);

               // Get end of transition.
              TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
               // Display information on floating date rule.
               if (!daylightEnd.IsFixedDateRule)
                  Console.WriteLine("      Ends at {0:t} on the {1} {2} of
{3}",
                                 daylightEnd.TimeOfDay,
                                 (WeekOfMonth) daylightEnd.Week,
```

```
                                 daylightEnd.DayOfWeek,
                                 dateInfo.GetMonthName(daylightEnd.Month));
               // Display information on fixed date rule.
               else
                  Console.WriteLine("      Ends at {0:t} on {1} {2}",
                                    daylightEnd.TimeOfDay,

dateInfo.GetMonthName(daylightEnd.Month),
                                    daylightEnd.Day);
            }
         }
      }
   }

   private enum WeekOfMonth
   {
      First = 1,
      Second = 2,
      Third = 3,
      Fourth = 4,
      Last = 5,
   }
}
// A portion of the output from the example might appear as follows:
//       Tonga Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 13:00:00
//             Supports DST: False
//             No adjustment rules defined.
//       Samoa Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 13:00:00
//             Supports DST: True
//             Adjustment Rules: 4
//          Adjustment rule from 1/1/0001 to 12/31/2009:
//             Delta: 00:00:00
//             Begins at 12:00 AM on January 1
//             Ends at 12:00 AM on January 1
//          Adjustment rule from 1/1/2010 to 12/31/2010:
//             Delta: 01:00:00
//             Begins at 11:59 PM on the Last Saturday of September
//             Ends at 12:00 AM on the First Friday of January
//          Adjustment rule from 1/1/2011 to 12/31/2011:
//             Delta: 01:00:00
//             Begins at 3:00 AM on the Fourth Saturday of September
//             Ends at 4:00 AM on the First Saturday of April
//          Adjustment rule from 1/1/2012 to 12/31/9999:
//             Delta: 01:00:00
//             Begins at 12:00 AM on the Last Sunday of September
//             Ends at 1:00 AM on the First Sunday of April
//       Line Islands Standard Time transition time information:
//          Time zone information:
//             Base UTC Offset: 14:00:00
//             Supports DST: False
//             No adjustment rules defined.
```

Remarks

The value of the DateStart property is a date value without a time component. It defines the date on which a particular adjustment rule goes into effect. This is the date in which a set of transitions (which typically are defined by one transition to daylight savings time and one transition back to standard time) go into effect. For example, an adjustment rule might go into effect on January 1, 2017, that provides for a transition to daylight savings time on the second Sunday of March and for a transition back to standard time on the first Sunday of November. Note that the starting date of the adjustment rule is not tied to the date of the first transition.

You can assign `DateTime.MinValue.Date` to the DateEnd property when you create a custom adjustment rule for use in a time zone-aware application that does not have to work with historic time zone information.

> **ⓘ Important**
>
> Unless there is a compelling reason to do otherwise, you should define the adjustment rule's start date to occur within the time interval during which the time zone observes standard time. Unless there is a compelling reason to do so, you should not define the adjustment rule's start date to occur within the time interval during which the time zone observes daylight saving time. For example, if a time zone's transition from daylight saving time occurs on the third Sunday of March and its transition to daylight saving time occurs on the first Sunday of October, the effective start date of the adjustment rule should not be January 1 of a particular year, since that date occurs within the period of daylight saving time.

By default, the registry in Windows XP defines a single adjustment rule whose start date is Monday, January 01, 0001 (the value of `DateTime.MinValue.Date`), for each time zone. For time zones in the United States, the registry in Windows Vista defines two adjustment rules:

- Monday, January 01, 0001, to Sunday, December 31, 2006.
- Monday, January 01, 2007, to Friday, December 31, 9999.

This means that, although time zone adjustment rules stored in the registry are useful for performing current time zone-related operations, they cannot be reliably used for retrieving historical time zone information. For information about defining a custom

time zone with multiple adjustment rules that can be used in a historical time zone-aware application, see How to: Create Time Zones with Adjustment Rules.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.AdjustmentRule.DaylightDelta Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the amount of time that is required to form the time zone's daylight saving time. This amount of time is added to the time zone's offset from Coordinated Universal Time (UTC).

```
C#
public TimeSpan DaylightDelta { get; }
```

Property Value

TimeSpan

A TimeSpan object that indicates the amount of time to add to the standard time changes as a result of the adjustment rule.

Remarks

The following formula defines a time zone's daylight saving time:

```
TimeZoneTime = BaseUtcOffset + DaylightDelta + UtcTime
```

The value of the DaylightDelta property can range from 14 hours to -14 hours.

> **Note**
>
> The **DaylightDelta** property measures the difference between the time zone's standard time and its daylight saving time. It does not apply to changes in a time zone's standard offset from Coordinated Universal Time (UTC). To represent a time

zone that has changed its standard time offset from UTC, you must call the **CreateCustomTimeZone** method to create a new time zone.

The most common value of the DaylightDelta property is 1.0 hours. The application of the daylight saving time adjustment rule increases the time zone's offset from Coordinated Universal Time (UTC) by one hour.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.AdjustmentRule.DaylightTransitionEnd Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets information about the annual transition from daylight saving time back to standard time.

C#

```
public TimeZoneInfo.TransitionTime DaylightTransitionEnd { get; }
```

Property Value

TimeZoneInfo.TransitionTime

A TimeZoneInfo.TransitionTime object that defines the annual transition from daylight saving time back to the time zone's standard time.

Remarks

The TimeZoneInfo.TransitionTime object returned by the DaylightTransitionEnd property defines either a fixed-date rule or a floating-date rule for the return to standard time. A fixed-date rule provides the month, day, and time of day on which the transition occurs. A floating-date rule provides the month, week, day of the week, and time of day on which the transition occurs.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule.DaylightTransitionStart Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets information about the annual transition from standard time to daylight saving time.

C#

```
public TimeZoneInfo.TransitionTime DaylightTransitionStart { get; }
```

Property Value

TimeZoneInfo.TransitionTime

A TimeZoneInfo.TransitionTime object that defines the annual transition from a time zone's standard time to daylight saving time.

Remarks

The TimeZoneInfo.TransitionTime object returned by the DaylightTransitionStart property defines either a fixed-date rule or a floating-date rule for the transition to daylight saving time. A fixed-date rule provides the month, day, and time of day on which the transition occurs. A floating-date rule provides the month, the week, the day of the week, and the time of day on which the transition occurs.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime)	Creates a new adjustment rule for a particular time zone.
CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime, TimeSpan)	Creates a new adjustment rule for a particular time zone.

CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime)

Creates a new adjustment rule for a particular time zone.

C#

```
public static TimeZoneInfo.AdjustmentRule CreateAdjustmentRule (DateTime
dateStart, DateTime dateEnd, TimeSpan daylightDelta,
TimeZoneInfo.TransitionTime daylightTransitionStart,
TimeZoneInfo.TransitionTime daylightTransitionEnd);
```

Parameters

`dateStart` DateTime

The effective date of the adjustment rule. If the value of the `dateStart` parameter is `DateTime.MinValue.Date`, this is the first adjustment rule in effect for a time zone.

`dateEnd` DateTime

The last date that the adjustment rule is in force. If the value of the `dateEnd` parameter is `DateTime.MaxValue.Date`, the adjustment rule has no end date.

`daylightDelta` TimeSpan

The time change that results from the adjustment. This value is added to the time zone's BaseUtcOffset property to obtain the correct daylight offset from Coordinated Universal Time (UTC). This value can range from -14 to 14.

`daylightTransitionStart` TimeZoneInfo.TransitionTime

An object that defines the start of daylight saving time.

`daylightTransitionEnd` TimeZoneInfo.TransitionTime

An object that defines the end of daylight saving time.

Returns

TimeZoneInfo.AdjustmentRule

An object that represents the new adjustment rule.

Exceptions

ArgumentException

The Kind property of the `dateStart` or `dateEnd` parameter does not equal Unspecified.

-or-

The `daylightTransitionStart` parameter is equal to the `daylightTransitionEnd` parameter.

-or-

The `dateStart` or `dateEnd` parameter includes a time of day value.

ArgumentOutOfRangeException

`dateEnd` is earlier than `dateStart`.

-or-

`daylightDelta` is less than -14 or greater than 14.

-or-

The Milliseconds property of the `daylightDelta` parameter is not equal to 0.

-or-

The Ticks property of the `daylightDelta` parameter does not equal a whole number of seconds.

Examples

The following example creates an alternate Central Standard Time zone and defines three adjustment rules for the periods 1976-1986, 1987-2006, and 2007 and beyond. These rules are added to a generic List<T> object whose elements are then copied to a TimeZoneInfo.AdjustmentRule array. This array is then used in the call to the TimeZoneInfo.CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[]) method.

C#

```
// Create alternate Central Standard Time to include historical time zone
information
//
// Declare necessary TimeZoneInfo.AdjustmentRule objects for time zone
TimeSpan delta = new TimeSpan(1, 0, 0);
TimeZoneInfo.AdjustmentRule adjustment;
List<TimeZoneInfo.AdjustmentRule> adjustmentList = new
List<TimeZoneInfo.AdjustmentRule>();
// Declare transition time variables to hold transition time information
TimeZoneInfo.TransitionTime transitionRuleStart, transitionRuleEnd;

// Define end rule (for 1976-2006)
transitionRuleEnd =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
2, 0, 0), 10, 5, DayOfWeek.Sunday);
// Define rule (1976-1986)
transitionRuleStart =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
2, 0, 0), 04, 05, DayOfWeek.Sunday);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1976, 1, 1), new DateTime(1986, 12, 31), delta,
transitionRuleStart, transitionRuleEnd);
adjustmentList.Add(adjustment);
// Define rule (1987-2006)
transitionRuleStart =
```

```
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
2, 0, 0), 04, 01, DayOfWeek.Sunday);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1987, 1, 1), new DateTime(2006, 12, 31), delta,
transitionRuleStart, transitionRuleEnd);
adjustmentList.Add(adjustment);
// Define rule (2007- )
transitionRuleStart =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
2, 0, 0), 03, 02, DayOfWeek.Sunday);
transitionRuleEnd =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
2, 0, 0), 11, 01, DayOfWeek.Sunday);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(2007, 01, 01), DateTime.MaxValue.Date, delta,
transitionRuleStart, transitionRuleEnd);
adjustmentList.Add(adjustment);

// Create custom U.S. Central Standard Time zone
TimeZoneInfo.CreateCustomTimeZone("Central Standard Time", new
TimeSpan(-6, 0, 0),
                "(GMT-06:00) Central Time (US Only)", "Central Standard
Time",
                "Central Daylight Time", adjustmentList.ToArray());
```

Remarks

You can use this method to create one or more adjustment rules for a custom time zone. An array of the TimeZoneInfo.AdjustmentRule objects returned by calls to this method can then be passed as the `adjustmentRules` parameter to two overloads of the CreateCustomTimeZone method. The example illustrates this procedure.

> **Note**
>
> The **CreateAdjustmentRule** method can be used only to define an adjustment rule for a new time zone; it cannot be used to modify an adjustment rule for an existing time zone.

The `dateStart` and `dateEnd` parameters must be date values without a time component or an ArgumentException is thrown. The time component can be removed by retrieving a DateTime value from the date and time's Date property as shown in the following statements:

```

DateTime.MinValue.Date
DateTime.MaxValue.Date
```

```
DateTime.Now.Date
dateVariable.Date
```

The Kind property of the `dateStart` and `dateEnd` parameters must also be DateTimeKind.Unspecified.

The value of the `daylightDelta` parameter can range from -14 to 14. The sum of the `daylightDelta` parameter and the `baseUtcOffset` parameter used in the call to the TimeZoneInfo.CreateCustomTimeZone method must also range from -14 to 14 or an InvalidTimeZoneException is thrown.

> **Note**
>
> The `daylightDelta` parameter defines the difference between a time zone's standard time and its daylight saving time. It is not intended to define the time zone's standard time offset from Coordinated Universal Time (UTC). The **TimeZoneInfo** class assumes that this offset from UTC is constant throughout the life of the time zone. To reflect a change in a time zone's offset from UTC that is not caused by the application of an adjustment rule, you must use the **CreateCustomTimeZone** method to create a new custom time zone.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CreateAdjustmentRule(DateTime, DateTime, TimeSpan, TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime, TimeSpan)

Creates a new adjustment rule for a particular time zone.

```
C#
```

```
public static TimeZoneInfo.AdjustmentRule CreateAdjustmentRule (DateTime
dateStart, DateTime dateEnd, TimeSpan daylightDelta,
TimeZoneInfo.TransitionTime daylightTransitionStart,
TimeZoneInfo.TransitionTime daylightTransitionEnd, TimeSpan
baseUtcOffsetDelta);
```

Parameters

`dateStart` DateTime

The effective date of the adjustment rule. If the value is `DateTime.MinValue.Date`, this is the first adjustment rule in effect for a time zone.

`dateEnd` DateTime

The last date that the adjustment rule is in force. If the value is `DateTime.MaxValue.Date`, the adjustment rule has no end date.

`daylightDelta` TimeSpan

The time change that results from the adjustment. This value is added to the time zone's BaseUtcOffset and BaseUtcOffsetDelta properties to obtain the correct daylight offset from Coordinated Universal Time (UTC). This value can range from -14 to 14.

`daylightTransitionStart` TimeZoneInfo.TransitionTime

The start of daylight saving time.

`daylightTransitionEnd` TimeZoneInfo.TransitionTime

The end of daylight saving time.

`baseUtcOffsetDelta` TimeSpan

The time difference with the base UTC offset for the time zone during the adjustment-rule period.

Returns

TimeZoneInfo.AdjustmentRule

The new adjustment rule.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Object)	Indicates whether the current instance is equal to another instance.
Equals(TimeZoneInfo+AdjustmentRule)	Determines whether the current TimeZoneInfo.AdjustmentRule object is equal to a second TimeZoneInfo.AdjustmentRule object.

Equals(Object)

Indicates whether the current instance is equal to another instance.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An instance to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the other instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Equals(TimeZoneInfo+AdjustmentRule)

Determines whether the current TimeZoneInfo.AdjustmentRule object is equal to a second TimeZoneInfo.AdjustmentRule object.

```
C#
public bool Equals (TimeZoneInfo.AdjustmentRule? other);
```

Parameters

`other` TimeZoneInfo.AdjustmentRule

The object to compare with the current object.

Returns

Boolean

`true` if both TimeZoneInfo.AdjustmentRule objects have equal values; otherwise, `false`.

Implements

Equals(T)

Examples

The following example calls the TimeZoneInfo.AdjustmentRule.Equals(TimeZoneInfo+AdjustmentRule) method to compare the adjustment rules for Central Standard Time with those for Canada Central Standard Time and Mexico Standard Time.

```
C#
```

```
   string timeZoneName = "";
   // Get CST, Canadian CST, and Mexican CST adjustment rules
   TimeZoneInfo.AdjustmentRule[] usCstAdjustments = null;
   TimeZoneInfo.AdjustmentRule[] canCstAdjustments = null;
   TimeZoneInfo.AdjustmentRule[] mexCstAdjustments = null;
   try
   {
      timeZoneName = "Central Standard Time";
      usCstAdjustments =
TimeZoneInfo.FindSystemTimeZoneById(timeZoneName).GetAdjustmentRules();
   }
   catch (TimeZoneNotFoundException)
   {
      Console.WriteLine("The {0} time zone is not defined in the
registry.",
                        timeZoneName);
   }
   catch (InvalidTimeZoneException)
   {
      Console.WriteLine("Data for the {0} time zone is invalid.",
                        timeZoneName);
   }
   try
   {
      timeZoneName = "Canada Central Standard Time";
      canCstAdjustments =
TimeZoneInfo.FindSystemTimeZoneById(timeZoneName).GetAdjustmentRules();
   }
   catch (TimeZoneNotFoundException)
   {
      Console.WriteLine("The {0} time zone is not defined in the
registry.",
                        timeZoneName);
   }
   catch (InvalidTimeZoneException)
   {
      Console.WriteLine("Data for the {0} time zone is invalid.",
                        timeZoneName);
   }
   try
   {
      timeZoneName = "Central Standard Time (Mexico)";
      mexCstAdjustments =
TimeZoneInfo.FindSystemTimeZoneById(timeZoneName).GetAdjustmentRules();
   }
   catch (TimeZoneNotFoundException)
   {
      Console.WriteLine("The {0} time zone is not defined in the
registry.",
                        timeZoneName);
   }
   catch (InvalidTimeZoneException)
   {
      Console.WriteLine("Data for the {0} time zone is invalid.",
```

```
                        timeZoneName);
    }
    // Determine if CST and other time zones have the same rules
    foreach(TimeZoneInfo.AdjustmentRule rule in usCstAdjustments)
    {
       Console.WriteLine("Comparing Central Standard Time rule for {0:d}
to {1:d} with:",
                        rule.DateStart, rule.DateEnd);
       // Compare with Canada Central Standard Time
       if (canCstAdjustments.Length == 0)
       {
          Console.WriteLine("   Canada Central Standard Time has no
adjustment rules.");
       }
       else
       {
          foreach (TimeZoneInfo.AdjustmentRule canRule in
canCstAdjustments)
          {
             Console.WriteLine("   Canadian CST for {0:d} to {1:d}: {2}",
                               canRule.DateStart, canRule.DateEnd,
                               rule.Equals(canRule) ? "Equal" : "Not
Equal");
          }
       }

       // Compare with Mexico Central Standard Time
       if (mexCstAdjustments.Length == 0)
       {
          Console.WriteLine("   Mexican Central Standard Time has no
adjustment rules.");
       }
       else
       {
          foreach (TimeZoneInfo.AdjustmentRule mexRule in
mexCstAdjustments)
          {
             Console.WriteLine("   Mexican CST for {0:d} to {1:d}: {2}",
                               mexRule.DateStart, mexRule.DateEnd,
                               rule.Equals(mexRule) ? "Equal" : "Not
Equal");
          }
       }
    }
    // This code displays the following output to the console:
    //
    // Comparing Central Standard Time rule for 1/1/0001 to 12/31/9999
with:
    //    Canada Central Standard Time has no adjustment rules.
    //    Mexican CST for 1/1/0001 to 12/31/9999: Equal
```

This code displays the following output to the console:

```
Comparing Central Standard Time rule for 1/1/0001 to 12/31/9999 with:
   Canada Central Standard Time has no adjustment rules.
   Mexican CST for 1/1/0001 to 12/31/9999: Equal
```

Remarks

To determine whether two TimeZoneInfo.AdjustmentRule objects are equal, the Equals(TimeZoneInfo+AdjustmentRule) method compares the member values of each object. Two adjustment rules are equal if they have the same effective dates, the same delta, and identical values for the TimeZoneInfo.TransitionTime objects returned by their DaylightTransitionStart and DaylightTransitionEnd properties.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.AdjustmentRule.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as a hash function for hashing algorithms and data structures such as hash tables.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that serves as the hash code for the current TimeZoneInfo.AdjustmentRule object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

TimeZoneInfo.AdjustmentRule.IDeserializationCallback.OnDeserialization Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Runs when the deserialization of a TimeZoneInfo.AdjustmentRule object is completed.

```
C#
void IDeserializationCallback.OnDeserialization (object sender);
```

Parameters

`sender` Object

The object that initiated the callback. The functionality for this parameter is not currently implemented.

Implements

OnDeserialization(Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.AdjustmentRule.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data that is required to serialize this object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with data.

`context` StreamingContext

The destination for this serialization (see StreamingContext).

Implements

GetObjectData(SerializationInfo, StreamingContext)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides information about a specific time change, such as the change from daylight saving time to standard time or vice versa, in a particular time zone.

C#

```
public readonly struct TimeZoneInfo.TransitionTime :
IEquatable<TimeZoneInfo.TransitionTime>,
System.Runtime.Serialization.IDeserializationCallback,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → TimeZoneInfo.TransitionTime

Implements IEquatable<TimeZoneInfo.TransitionTime> , IDeserializationCallback , ISerializable

Remarks

You can use the TimeZoneInfo.TransitionTime structure to indicate when a transition from standard time to daylight saving time, or from daylight saving time back to standard time, occurs. This structure supports both fixed-date rules and floating-date rules. Use fixed-date rules for time transitions that occur on a specific day of a specific month (such as 2:00 A.M. on November 3). Use floating-date rules for time transitions that occur on a specific day of a specific week of a specific month (such as 2:00 A.M. on the first Sunday of November).

The following table compares the properties used in fixed-date and floating-date transitions:

Expand table

	Month	Week	Day	Time
Fixed-date rule	Month	N/A	Day	TimeOfDay
Floating-date rule	Month	Week	DayOfWeek	TimeOfDay

For both fixed-date and floating-date transitions, the TimeOfDay property gets the time at which the time change occurs. For transitions from standard to daylight saving time, this is the time zone's standard time value. For transitions from daylight saving to standard time, this is the time zone's daylight saving time value. This is a DateTime value whose date component is ignored; its year, month, and day value must always equal 1.

> **Note**
>
> An instance of the **TimeZoneInfo.TransitionTime** structure is immutable. Once an object has been created, its values cannot be modified.

A TimeZoneInfo.TransitionTime object can be created by calling the `static` (`Shared` in Visual Basic) CreateFixedDateRule and CreateFloatingDateRule methods to create a fixed or floating-date rule, respectively. The starting and ending TimeZoneInfo.TransitionTime objects are then supplied as parameters to the CreateAdjustmentRule method to create a new adjustment rule that includes this transition time information.

The DaylightTransitionStart and DaylightTransitionEnd properties of an TimeZoneInfo.AdjustmentRule object return a TimeZoneInfo.TransitionTime object.

Properties

Expand table

Day	Gets the day on which the time change occurs.
DayOfWeek	Gets the day of the week on which the time change occurs.
IsFixedDateRule	Gets a value indicating whether the time change occurs at a fixed date and time (such as November 1) or a floating date and time (such as the last Sunday of October).
Month	Gets the month in which the time change occurs.
TimeOfDay	Gets the hour, minute, and second at which the time change occurs.
Week	Gets the week of the month in which a time change occurs.

Methods

Expand table

CreateFixedDateRule(DateTime, Int32, Int32)	Defines a time change that uses a fixed-date rule (that is, a time change that occurs on a specific day of a specific month).
CreateFloatingDateRule(DateTime, Int32, Int32, DayOfWeek)	Defines a time change that uses a floating-date rule (that is, a time change that occurs on a specific day of a specific week of a specific month).
Equals(Object)	Determines whether an object has identical values to the current TimeZoneInfo.TransitionTime object.
Equals(TimeZoneInfo+TransitionTime)	Determines whether the current TimeZoneInfo.TransitionTime object has identical values to a second TimeZoneInfo.TransitionTime object.
GetHashCode()	Serves as a hash function for hashing algorithms and data structures such as hash tables.

Operators

Expand table

Equality(TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime)	Determines whether two specified TimeZoneInfo.TransitionTime objects are equal.
Inequality(TimeZoneInfo+TransitionTime, TimeZoneInfo+TransitionTime)	Determines whether two specified TimeZoneInfo.TransitionTime objects are not equal.

Explicit Interface Implementations

Expand table

IDeserializationCallback.OnDeserialization(Object)	Runs when the deserialization of an object has been completed.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data that is required to serialize this object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.TransitionTime.Day Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day on which the time change occurs.

C#

```
public int Day { get; }
```

Property Value

Int32

The day on which the time change occurs.

Examples

The following example enumerates the time zones found on the local computer and displays transition time information for all fixed-date transitions. The time information includes the day on which the time change occurs.

C#

```
private void GetFixedTransitionTimes()
{
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   DateTimeFormatInfo dateInfo = CultureInfo.CurrentCulture.DateTimeFormat;
   foreach (TimeZoneInfo zone in timeZones)
   {
      TimeZoneInfo.AdjustmentRule[] adjustmentRules =
zone.GetAdjustmentRules();
      foreach (TimeZoneInfo.AdjustmentRule adjustmentRule in
adjustmentRules)
      {
         TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
```

```
         if (daylightStart.IsFixedDateRule)
            Console.WriteLine("For {0}, daylight savings time begins at
{1:t} on {2} {3} from {4:d} to {5:d}.",
                              zone.StandardName,
                              daylightStart.TimeOfDay,
                              dateInfo.GetMonthName(daylightStart.Month),
                              daylightStart.Day,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd);
         TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
         if (daylightEnd.IsFixedDateRule)
            Console.WriteLine("For {0}, daylight savings time ends at {1:t}
on {2} {3} from {4:d} to {5:d}.",
                              zone.StandardName,
                              daylightEnd.TimeOfDay,
                              dateInfo.GetMonthName(daylightEnd.Month),
                              daylightEnd.Day,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd);
      }
   }
}
```

Remarks

The Day property returns a valid value only if the IsFixedDateRule property is `true`.

The Day property value corresponds to the value of the `day` parameter of the CreateFixedDateRule method. If its value is greater than the number of days in the month of the transition, the transition occurs on the last day of the month.

The Day property indicates the day of the month on which a fixed-date rule is applied (for example, April 15). In contrast, the DayOfWeek property indicates the day of the week on which a floating-date rule is applied (for example, the second Sunday of November).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.DayOfWeek Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the day of the week on which the time change occurs.

C#

```
public DayOfWeek DayOfWeek { get; }
```

Property Value

DayOfWeek

The day of the week on which the time change occurs.

Examples

The following example enumerates the time zones found on the local computer and displays transition time information for all floating-date transitions. The time information includes the day of the week on which the time change occurs.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
   Last = 5
}

private void GetFloatingTransitionTimes()
{
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   foreach (TimeZoneInfo zone in timeZones)
```

```
    {
        TimeZoneInfo.AdjustmentRule[] adjustmentRules =
zone.GetAdjustmentRules();
        DateTimeFormatInfo dateInfo =
CultureInfo.CurrentCulture.DateTimeFormat;
        foreach (TimeZoneInfo.AdjustmentRule adjustmentRule in
adjustmentRules)
        {
            TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
            if (! daylightStart.IsFixedDateRule)
                Console.WriteLine("{0}, {1:d}-{2:d}: Begins at {3:t} on the {4}
{5} of {6}.",
                                  zone.StandardName,
                                  adjustmentRule.DateStart,
                                  adjustmentRule.DateEnd,
                                  daylightStart.TimeOfDay,
                                  ((WeekOfMonth)daylightStart.Week).ToString(),
                                  daylightStart.DayOfWeek.ToString(),
                                  dateInfo.GetMonthName(daylightStart.Month));

            TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
            if (! daylightEnd.IsFixedDateRule)
                Console.WriteLine("{0}, {1:d}-{2:d}: Ends at {3:t} on the {4}
{5} of {6}.",
                                  zone.StandardName,
                                  adjustmentRule.DateStart,
                                  adjustmentRule.DateEnd,
                                  daylightEnd.TimeOfDay,
                                  ((WeekOfMonth)daylightEnd.Week).ToString(),
                                  daylightEnd.DayOfWeek.ToString(),
                                  dateInfo.GetMonthName(daylightEnd.Month));
        }
    }
}
```

Remarks

The DayOfWeek property returns a valid value only if the IsFixedDateRule property is `false`.

The DayOfWeek property indicates the day of the week on which a floating-date rule is applied (for example, the second Sunday of November). In contrast, the Day property indicates the day of the month on which a fixed-date rule is applied (for example, April 15).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.IsFixedDateRule Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the time change occurs at a fixed date and time (such as November 1) or a floating date and time (such as the last Sunday of October).

C#

```
public bool IsFixedDateRule { get; }
```

Property Value

Boolean

`true` if the time change rule is fixed-date; `false` if the time change rule is floating-date.

Examples

The following example lists the transition times to and from daylight saving time for all time zones that are available on the local system. For time zones with fixed-date rules, it displays transition time information from the properties of the TimeZoneInfo.TransitionTime object. For time zones without fixed-date rules, it uses a Calendar object that represents the current system calendar to determine the actual start and end dates of the transition. The example displays the results to the console.

C#

```
private void GetTransitionTimes(int year)
{
   // Instantiate DateTimeFormatInfo object for month names
   DateTimeFormatInfo dateFormat =
CultureInfo.CurrentCulture.DateTimeFormat;

   // Get and iterate time zones on local computer
   ReadOnlyCollection<TimeZoneInfo> timeZones =
```

```
TimeZoneInfo.GetSystemTimeZones();
   foreach (TimeZoneInfo timeZone in timeZones)
   {
      Console.WriteLine("{0}:", timeZone.StandardName);
      TimeZoneInfo.AdjustmentRule[] adjustments =
timeZone.GetAdjustmentRules();
      int startYear = year;
      int endYear = startYear;

      if (adjustments.Length == 0)
      {
         Console.WriteLine("   No adjustment rules.");
      }
      else
      {
         TimeZoneInfo.AdjustmentRule adjustment = GetAdjustment(adjustments,
year);
         if (adjustment == null)
         {
            Console.WriteLine("   No adjustment rules available for this
year.");
            continue;
         }
         TimeZoneInfo.TransitionTime startTransition, endTransition;

         // Determine if starting transition is fixed
         startTransition = adjustment.DaylightTransitionStart;
         // Determine if starting transition is fixed and display transition
info for year
         if (startTransition.IsFixedDateRule)
            Console.WriteLine("   Begins on {0} {1} at {2:t}",

dateFormat.GetMonthName(startTransition.Month),
                              startTransition.Day,
                              startTransition.TimeOfDay);
         else
            DisplayTransitionInfo(startTransition, startYear, "Begins on");

         // Determine if ending transition is fixed and display transition
info for year
         endTransition = adjustment.DaylightTransitionEnd;

         // Does the transition back occur in an earlier month (i.e.,
         // the following year) than the transition to DST? If so, make
         // sure we have the right adjustment rule.
         if (endTransition.Month < startTransition.Month)
         {
            endTransition = GetAdjustment(adjustments, year +
1).DaylightTransitionEnd;
            endYear++;
         }

         if (endTransition.IsFixedDateRule)
            Console.WriteLine("   Ends on {0} {1} at {2:t}",
                              dateFormat.GetMonthName(endTransition.Month),
```

```
                                endTransition.Day,
                                endTransition.TimeOfDay);
         else
            DisplayTransitionInfo(endTransition, endYear, "Ends on");
      }
   }
}

private static TimeZoneInfo.AdjustmentRule
GetAdjustment(TimeZoneInfo.AdjustmentRule[] adjustments,
                                                    int year)
{
   // Iterate adjustment rules for time zone
   foreach (TimeZoneInfo.AdjustmentRule adjustment in adjustments)
   {
      // Determine if this adjustment rule covers year desired
      if (adjustment.DateStart.Year <= year && adjustment.DateEnd.Year >=
year)
         return adjustment;
   }
   return null;
}

private void DisplayTransitionInfo(TimeZoneInfo.TransitionTime transition,
int year, string label)
{
   // For non-fixed date rules, get local calendar
   Calendar cal = CultureInfo.CurrentCulture.Calendar;
   // Get first day of week for transition
   // For example, the 3rd week starts no earlier than the 15th of the month
   int startOfWeek = transition.Week * 7 - 6;
   // What day of the week does the month start on?
   int firstDayOfWeek = (int) cal.GetDayOfWeek(new DateTime(year,
transition.Month, 1));
   // Determine how much start date has to be adjusted
   int transitionDay;
   int changeDayOfWeek = (int) transition.DayOfWeek;

   if (firstDayOfWeek <= changeDayOfWeek)
      transitionDay = startOfWeek + (changeDayOfWeek - firstDayOfWeek);
   else
      transitionDay = startOfWeek + (7 - firstDayOfWeek + changeDayOfWeek);

   // Adjust for months with no fifth week
   if (transitionDay > cal.GetDaysInMonth(year, transition.Month))
      transitionDay -= 7;

   Console.WriteLine("   {0} {1}, {2:d} at {3:t}",
                     label,
                     transition.DayOfWeek,
                     new DateTime(year, transition.Month, transitionDay),
                     transition.TimeOfDay);
}
```

Remarks

A fixed-date rule indicates that the transition occurs on the same date and time of each year to which the adjustment rule applies. For example, a time change that occurs every November 3 follows a fixed-date rule. A floating-date rule indicates that the transition occurs on a specific day of a specific week of a specific month for each year to which the adjustment rule applies. For example, a time change that occurs on the first Sunday of November follows a floating-date rule.

The value of the IsFixedDateRule property determines which properties of a TimeZoneInfo.TransitionTime object have valid values. The following table indicates which properties are affected by the value of the IsFixedDateRule property.

Expand table

TransitionTime property	IsFixedDateRule = true	IsFixedDateRule = false
`Day`	Valid	Unused
`DayOfWeek`	Unused	Valid
`Week`	Unused	Valid
`Month`	Valid	Valid
`TimeOfDay`	Valid	Valid

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeZoneInfo.TransitionTime.Month Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the month in which the time change occurs.

C#

```
public int Month { get; }
```

Property Value

Int32

The month in which the time change occurs.

Examples

The following example enumerates the time zones found on the local computer and displays transition time information for all floating-date transitions. The time information includes the month in which the time change occurs.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
   Last = 5
}

private void GetFloatingTransitionTimes()
{
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   foreach (TimeZoneInfo zone in timeZones)
```

```
   {
      TimeZoneInfo.AdjustmentRule[] adjustmentRules =
zone.GetAdjustmentRules();
      DateTimeFormatInfo dateInfo =
CultureInfo.CurrentCulture.DateTimeFormat;
      foreach (TimeZoneInfo.AdjustmentRule adjustmentRule in
adjustmentRules)
      {
         TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
         if (! daylightStart.IsFixedDateRule)
            Console.WriteLine("{0}, {1:d}-{2:d}: Begins at {3:t} on the {4}
{5} of {6}.",
                              zone.StandardName,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd,
                              daylightStart.TimeOfDay,
                              ((WeekOfMonth)daylightStart.Week).ToString(),
                              daylightStart.DayOfWeek.ToString(),
                              dateInfo.GetMonthName(daylightStart.Month));

         TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
         if (! daylightEnd.IsFixedDateRule)
            Console.WriteLine("{0}, {1:d}-{2:d}: Ends at {3:t} on the {4}
{5} of {6}.",
                              zone.StandardName,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd,
                              daylightEnd.TimeOfDay,
                              ((WeekOfMonth)daylightEnd.Week).ToString(),
                              daylightEnd.DayOfWeek.ToString(),
                              dateInfo.GetMonthName(daylightEnd.Month));
      }
   }
}
```

Remarks

Valid values for the Month property range from 1 to 12.

The Month property is used for both fixed-date and floating-date rules.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.TimeOfDay Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hour, minute, and second at which the time change occurs.

C#

```
public DateTime TimeOfDay { get; }
```

Property Value

DateTime

The time of day at which the time change occurs.

Examples

The following example enumerates the time zones found on the local computer and displays transition time information for those time zones that have adjustment rules. The time information includes the time of day at which the time change occurs.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
   Last = 5,
}

public void GetAllTransitionTimes()
{
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   DateTimeFormatInfo dateInfo = CultureInfo.CurrentCulture.DateTimeFormat;
```

```
   foreach (TimeZoneInfo zone in timeZones)
   {
      Console.WriteLine("{0} transition time information:",
zone.StandardName);
      TimeZoneInfo.AdjustmentRule[] adjustmentRules=
zone.GetAdjustmentRules();

      // Indicate that time zone has no adjustment rules
      if (adjustmentRules.Length == 0)
      {
         Console.WriteLine("   No adjustment rules defined.");
      }
      else
      {
         // Iterate adjustment rules
         foreach (TimeZoneInfo.AdjustmentRule adjustmentRule in
adjustmentRules)
         {
            Console.WriteLine("   Adjustment rule from {0:d} to {1:d}:",
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd);

            // Get start of transition
            TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
            // Display information on fixed date rule
            if (! daylightStart.IsFixedDateRule)
               Console.WriteLine("      Begins at {0:t} on the {1} {2} of
{3}.",
                              daylightStart.TimeOfDay,
                              ((WeekOfMonth)daylightStart.Week).ToString(),
                              daylightStart.DayOfWeek.ToString(),
                              dateInfo.GetMonthName(daylightStart.Month));
            // Display information on floating date rule
            else
               Console.WriteLine("      Begins at {0:t} on the {1} {2} of
{3}.",
                                 daylightStart.TimeOfDay,

((WeekOfMonth)daylightStart.Week).ToString(),
                                 daylightStart.DayOfWeek.ToString(),

dateInfo.GetMonthName(daylightStart.Month));

            // Get end of transition
            TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
            // Display information on fixed date rule
            if (! daylightEnd.IsFixedDateRule)
               Console.WriteLine("      Ends at {0:t} on the {1} {2} of
{3}.",
                              daylightEnd.TimeOfDay,
                              ((WeekOfMonth)daylightEnd.Week).ToString(),
                              daylightEnd.DayOfWeek.ToString(),
```

```
                                dateInfo.GetMonthName(daylightEnd.Month));
            // Display information on floating date rule
            else
               Console.WriteLine("      Ends at {0:t} on the {1} {2} of
{3}.",
                                  daylightStart.TimeOfDay,

((WeekOfMonth)daylightStart.Week).ToString(),
                                  daylightStart.DayOfWeek.ToString(),

dateInfo.GetMonthName(daylightStart.Month));
         }
      }
   }
}
```

Remarks

For transitions from standard time to daylight saving time, the TimeOfDay value represents the time of the transition in the time zone's standard time. For transitions from daylight saving time to standard time, it represents the time of the transition in the time zone's daylight saving time.

The TimeOfDay property defines only the time of a time change, but not its date. The date is determined by the Month and Day properties for fixed-rule changes, and by the Month, Week, and DayOfWeek properties for floating-rule changes. The date component of this DateTime value is ignored; the value of the year, month, and day is always 1.

The TimeOfDay property is used for both fixed-date and floating-date transitions.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.Week Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the week of the month in which a time change occurs.

C#

```
public int Week { get; }
```

Property Value

Int32

The week of the month in which the time change occurs.

Examples

The following example enumerates the time zones found on the local computer and displays transition time information for all floating-date transitions. The time information includes the week of the month in which the time change occurs. The example also defines a `WeekOfMonth` enumeration to display a string that represents the ordinal value of the Week property rather than an integer.

C#

```
private enum WeekOfMonth
{
   First = 1,
   Second = 2,
   Third = 3,
   Fourth = 4,
   Last = 5
}

private void GetFloatingTransitionTimes()
{
```

```
   ReadOnlyCollection<TimeZoneInfo> timeZones =
TimeZoneInfo.GetSystemTimeZones();
   foreach (TimeZoneInfo zone in timeZones)
   {
      TimeZoneInfo.AdjustmentRule[] adjustmentRules =
zone.GetAdjustmentRules();
      DateTimeFormatInfo dateInfo =
CultureInfo.CurrentCulture.DateTimeFormat;
      foreach (TimeZoneInfo.AdjustmentRule adjustmentRule in
adjustmentRules)
      {
         TimeZoneInfo.TransitionTime daylightStart =
adjustmentRule.DaylightTransitionStart;
         if (! daylightStart.IsFixedDateRule)
            Console.WriteLine("{0}, {1:d}-{2:d}: Begins at {3:t} on the {4}
{5} of {6}.",
                              zone.StandardName,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd,
                              daylightStart.TimeOfDay,
                              ((WeekOfMonth)daylightStart.Week).ToString(),
                              daylightStart.DayOfWeek.ToString(),
                              dateInfo.GetMonthName(daylightStart.Month));

         TimeZoneInfo.TransitionTime daylightEnd =
adjustmentRule.DaylightTransitionEnd;
         if (! daylightEnd.IsFixedDateRule)
            Console.WriteLine("{0}, {1:d}-{2:d}: Ends at {3:t} on the {4}
{5} of {6}.",
                              zone.StandardName,
                              adjustmentRule.DateStart,
                              adjustmentRule.DateEnd,
                              daylightEnd.TimeOfDay,
                              ((WeekOfMonth)daylightEnd.Week).ToString(),
                              daylightEnd.DayOfWeek.ToString(),
                              dateInfo.GetMonthName(daylightEnd.Month));
      }
   }
}
```

Remarks

The value of the Week property is used only for time changes with floating-date rules. Valid values can range from 1 to 5.

The Month property defines the month in which the time change occurs. The Week property determines the week on which the transition occurs. The DayOfWeek property defines the day of the week on which the transition occurs. The value of the Week property is determined as shown in the following table.

Expand table

If the Week property value is	The transition occurs on
1	The first occurrence of the DayOfWeek value in Month.
2	The second occurrence of the DayOfWeek value in Month.
3	The third occurrence of the DayOfWeek value in Month.
4	The fourth occurrence of the DayOfWeek value in Month.
5	The last occurrence of the DayOfWeek value in Month.

For example, if a transition occurs on the first Sunday of March, the value of the Week property is 1. If it occurs on the last Sunday of March, the value of the Week property is 5.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Day
- Month

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.TransitionTime.CreateFixedDateRule Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a time change that uses a fixed-date rule (that is, a time change that occurs on a specific day of a specific month).

C#

```
public static TimeZoneInfo.TransitionTime CreateFixedDateRule (DateTime
timeOfDay, int month, int day);
```

Parameters

timeOfDay DateTime

The time at which the time change occurs. This parameter corresponds to the TimeOfDay property.

month Int32

The month in which the time change occurs. This parameter corresponds to the Month property.

day Int32

The day of the month on which the time change occurs. This parameter corresponds to the Day property.

Returns

TimeZoneInfo.TransitionTime

Data about the time change.

Exceptions

ArgumentException

The `timeOfDay` parameter has a non-default date component.

-or-

The `timeOfDay` parameter's Kind property is not Unspecified.

-or-

The `timeOfDay` parameter does not represent a whole number of milliseconds.

ArgumentOutOfRangeException

The `month` parameter is less than 1 or greater than 12.

-or-

The `day` parameter is less than 1 or greater than 31.

Examples

The following example creates time zone transitions for an imaginary time zone by using both a fixed-date rule and a floating-date rule. The fixed-date rule defines a time zone adjustment that starts at 2:00 A.M. on March 15 and ends at 3:00 A.M. on November 15 for the years 1900 to 1955.

C#

```
// Declare necessary TimeZoneInfo.AdjustmentRule objects for time zone
TimeZoneInfo imaginaryTZ;
TimeSpan delta = new TimeSpan(1, 0, 0);
TimeZoneInfo.AdjustmentRule adjustment;
List<TimeZoneInfo.AdjustmentRule> adjustmentList = new
List<TimeZoneInfo.AdjustmentRule>();
// Declare transition time variables to hold transition time information
TimeZoneInfo.TransitionTime transitionRuleStart, transitionRuleEnd;

// Define a fictitious new time zone consisting of fixed and floating
adjustment rules
// Define fixed rule (for 1900-1955)
transitionRuleStart = TimeZoneInfo.TransitionTime.CreateFixedDateRule(new
DateTime(1, 1, 1, 2, 0, 0), 3, 15);
transitionRuleEnd = TimeZoneInfo.TransitionTime.CreateFixedDateRule(new
DateTime(1, 1, 1, 3, 0, 0), 11, 15);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1900, 1, 1), new DateTime(1955, 12, 31),
             delta, transitionRuleStart, transitionRuleEnd);
adjustmentList.Add(adjustment);
// Define floating rule (for 1956- )
transitionRuleStart = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
```

```
DateTime(1, 1, 1, 2, 0, 0), 3, 5, DayOfWeek.Sunday);
transitionRuleEnd = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 3, 0, 0), 10, 4, DayOfWeek.Sunday);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1956, 1, 1), DateTime.MaxValue.Date, delta, transitionRuleStart,
transitionRuleEnd);
adjustmentList.Add(adjustment);

// Create fictitious time zone
imaginaryTZ = TimeZoneInfo.CreateCustomTimeZone("Fictitious Standard Time",
new TimeSpan(-9, 0, 0),
                "(GMT-09:00) Fictitious Time", "Fictitious Standard Time",
                "Fictitious Daylight Time", adjustmentList.ToArray());
```

Remarks

This method creates a fixed-date rule (that is, a time change that occurs on a specific day of a specific month). For example, a time change that always occurs on October 28 follows a fixed-date rule.

For transitions from standard time to daylight saving time, the `timeOfDay` argument represents the time of the transition in the time zone's standard time. For transitions from daylight saving time to standard time, it represents the time of the transition in the time zone's daylight saving time. Note that this is a DateTime value whose year, month, and date values must all equal 1.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TimeZoneInfo.TransitionTime.CreateFloatingDateRule Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines a time change that uses a floating-date rule (that is, a time change that occurs on a specific day of a specific week of a specific month).

C#

```
public static TimeZoneInfo.TransitionTime CreateFloatingDateRule (DateTime
timeOfDay, int month, int week, DayOfWeek dayOfWeek);
```

Parameters

timeOfDay DateTime

The time at which the time change occurs. This parameter corresponds to the TimeOfDay property.

month Int32

The month in which the time change occurs. This parameter corresponds to the Month property.

week Int32

The week of the month in which the time change occurs. Its value can range from 1 to 5, with 5 representing the last week of the month. This parameter corresponds to the Week property.

dayOfWeek DayOfWeek

The day of the week on which the time change occurs. This parameter corresponds to the DayOfWeek property.

Returns

TimeZoneInfo.TransitionTime

Data about the time change.

Exceptions

ArgumentException

The `timeOfDay` parameter has a non-default date component.

-or-

The `timeOfDay` parameter does not represent a whole number of milliseconds.

-or-

The `timeOfDay` parameter's Kind property is not Unspecified.

ArgumentOutOfRangeException

`month` is less than 1 or greater than 12.

-or-

`week` is less than 1 or greater than 5.

-or-

The `dayOfWeek` parameter is not a member of the DayOfWeek enumeration.

Examples

The following example creates time zone transitions for an imaginary time zone by using both a fixed-date rule and a floating-date rule. The floating-date rule defines a time zone adjustment that starts at 2:00 A.M. on the last Sunday of March and ends at 3:00 A.M. on the fourth Sunday in October for the years 1956 onward.

C#

```
// Declare necessary TimeZoneInfo.AdjustmentRule objects for time zone
TimeZoneInfo imaginaryTZ;
TimeSpan delta = new TimeSpan(1, 0, 0);
TimeZoneInfo.AdjustmentRule adjustment;
List<TimeZoneInfo.AdjustmentRule> adjustmentList = new
List<TimeZoneInfo.AdjustmentRule>();
// Declare transition time variables to hold transition time information
TimeZoneInfo.TransitionTime transitionRuleStart, transitionRuleEnd;

// Define a fictitious new time zone consisting of fixed and floating
adjustment rules
```

```
// Define fixed rule (for 1900-1955)
transitionRuleStart = TimeZoneInfo.TransitionTime.CreateFixedDateRule(new
DateTime(1, 1, 1, 2, 0, 0), 3, 15);
transitionRuleEnd = TimeZoneInfo.TransitionTime.CreateFixedDateRule(new
DateTime(1, 1, 1, 3, 0, 0), 11, 15);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1900, 1, 1), new DateTime(1955, 12, 31),
             delta, transitionRuleStart, transitionRuleEnd);
adjustmentList.Add(adjustment);
// Define floating rule (for 1956- )
transitionRuleStart = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 2, 0, 0), 3, 5, DayOfWeek.Sunday);
transitionRuleEnd = TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new
DateTime(1, 1, 1, 3, 0, 0), 10, 4, DayOfWeek.Sunday);
adjustment = TimeZoneInfo.AdjustmentRule.CreateAdjustmentRule(new
DateTime(1956, 1, 1), DateTime.MaxValue.Date, delta, transitionRuleStart,
transitionRuleEnd);
adjustmentList.Add(adjustment);

// Create fictitious time zone
imaginaryTZ = TimeZoneInfo.CreateCustomTimeZone("Fictitious Standard Time",
new TimeSpan(-9, 0, 0),
                "(GMT-09:00) Fictitious Time", "Fictitious Standard Time",
                "Fictitious Daylight Time", adjustmentList.ToArray());
```

Remarks

This method creates a floating-date rule (that is, a time change that occurs on a specific day of a specific week of a specific month). For example, a time change that occurs on the last Sunday of October follows a floating-date rule.

For transitions from standard time to daylight saving time, the `timeOfDay` argument represents the time of the transition in the time zone's standard time. For transitions from daylight saving time to standard time, it represents the time of the transition in the time zone's daylight saving time. Note that this is a DateTime value whose year, month, and date values must all equal 1.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two TimeZoneInfo.TransitionTime objects have identical values.

Overloads

Expand table

Equals(Object)	Determines whether an object has identical values to the current TimeZoneInfo.TransitionTime object.
Equals(TimeZoneInfo+TransitionTime)	Determines whether the current TimeZoneInfo.TransitionTime object has identical values to a second TimeZoneInfo.TransitionTime object.

Equals(Object)

Determines whether an object has identical values to the current TimeZoneInfo.TransitionTime object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with the current TimeZoneInfo.TransitionTime object.

Returns

Boolean

`true` if the two objects are equal; otherwise, `false`.

Examples

The following example illustrates calls to the Equals method.

```C#
TimeZoneInfo.TransitionTime tt1 =
TimeZoneInfo.TransitionTime.CreateFixedDateRule(new DateTime(1, 1, 1, 02,
00, 00), 11, 03);
TimeZoneInfo.TransitionTime tt2 =
TimeZoneInfo.TransitionTime.CreateFixedDateRule(new DateTime(1, 1, 1, 02,
00, 00), 11, 03);
TimeZoneInfo.TransitionTime tt3 =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
02, 00, 00), 10, 05, DayOfWeek.Sunday);
TimeZoneInfo tz = TimeZoneInfo.Local;
Console.WriteLine(tt1.Equals(tz));         // Returns False (overload
with argument of type Object)
Console.WriteLine(tt1.Equals(tt1));        // Returns True (an object
always equals itself)
Console.WriteLine(tt1.Equals(tt2));        // Returns True (identical
property values)
Console.WriteLine(tt1.Equals(tt3));        // Returns False (different
property values)
```

Remarks

Equality is defined as follows: The `obj` parameter must be a TimeZoneInfo.TransitionTime object and must have property values identical to the current TimeZoneInfo.TransitionTime object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(TimeZoneInfo+TransitionTime)

Determines whether the current TimeZoneInfo.TransitionTime object has identical values to a second TimeZoneInfo.TransitionTime object.

```
C#

public bool Equals (TimeZoneInfo.TransitionTime other);
```

Parameters

`other` TimeZoneInfo.TransitionTime

An object to compare to the current instance.

Returns

Boolean

`true` if the two objects have identical property values; otherwise, `false`.

Implements

Equals(T)

Examples

The following example illustrates calls to the Equals method.

```
C#

TimeZoneInfo.TransitionTime tt1 =
TimeZoneInfo.TransitionTime.CreateFixedDateRule(new DateTime(1, 1, 1, 02,
00, 00), 11, 03);
TimeZoneInfo.TransitionTime tt2 =
TimeZoneInfo.TransitionTime.CreateFixedDateRule(new DateTime(1, 1, 1, 02,
00, 00), 11, 03);
TimeZoneInfo.TransitionTime tt3 =
TimeZoneInfo.TransitionTime.CreateFloatingDateRule(new DateTime(1, 1, 1,
02, 00, 00), 10, 05, DayOfWeek.Sunday);
Console.WriteLine(tt1.Equals(tt1));        // Returns True (an object
always equals itself)
Console.WriteLine(tt1.Equals(tt2));        // Returns True (identical
property values)
Console.WriteLine(tt1.Equals(tt3));        // Returns False (different
property values)
```

Remarks

To determine whether the two TimeZoneInfo.TransitionTime objects are equal, this method performs a property-by-property comparison. The method returns `true` only if the current TimeZoneInfo.TransitionTime object and the `other` object have identical values for each of the TimeZoneInfo.TransitionTime object's six property values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TimeZoneInfo.TransitionTime.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as a hash function for hashing algorithms and data structures such as hash tables.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that serves as the hash code for this TimeZoneInfo.TransitionTime object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

TimeZoneInfo.TransitionTime.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified TimeZoneInfo.TransitionTime objects are equal.

C#

```
public static bool operator == (TimeZoneInfo.TransitionTime t1,
TimeZoneInfo.TransitionTime t2);
```

Parameters

t1 TimeZoneInfo.TransitionTime

The first object to compare.

t2 TimeZoneInfo.TransitionTime

The second object to compare.

Returns

Boolean

`true` if `t1` and `t2` have identical values; otherwise, `false`.

Remarks

Two TimeZoneInfo.TransitionTime objects are equal if they have identical values for each of their six properties. Otherwise, they are unequal.

The equivalent method for this operator is TimeZoneInfo.TransitionTime.Equals(TimeZoneInfo+TransitionTime).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified TimeZoneInfo.TransitionTime objects are not equal.

C#

```
public static bool operator != (TimeZoneInfo.TransitionTime t1,
TimeZoneInfo.TransitionTime t2);
```

Parameters

t1 TimeZoneInfo.TransitionTime

The first object to compare.

t2 TimeZoneInfo.TransitionTime

The second object to compare.

Returns

Boolean

`true` if `t1` and `t2` have any different member values; otherwise, `false`.

Remarks

Two TimeZoneInfo.TransitionTime objects are unequal if they have different values for any of their six properties. Otherwise, they are equal.

The equivalent method for this operator is TimeZoneInfo.TransitionTime.Equals(TimeZoneInfo+TransitionTime).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.IDeserializationCallback.OnDeserialization Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Runs when the deserialization of an object has been completed.

C#

```
void IDeserializationCallback.OnDeserialization (object sender);
```

Parameters

`sender` Object

The object that initiated the callback. The functionality for this parameter is not currently implemented.

Implements

OnDeserialization(Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneInfo.TransitionTime.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data that is required to serialize this object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object to populate with data.

`context` StreamingContext

The destination for this serialization (see StreamingContext).

Implements

GetObjectData(SerializationInfo, StreamingContext)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneNotFoundException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a time zone cannot be found.

C#

```
public class TimeZoneNotFoundException : Exception
```

Inheritance Object → Exception → TimeZoneNotFoundException

Remarks

This exception is thrown by the ConvertTimeBySystemTimeZoneId and FindSystemTimeZoneById methods when a time zone identifier cannot be found on the local system, or when there is no data associated with a particular time zone identifier.

Because the registry serves as the repository of time zone information in Windows XP and Windows Vista, this exception indicates that the registry contains no information about a particular time zone. Time zone information is stored in the subkeys of HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Time Zones.

If an application depends on the presence of a particular time zone and the attempt to retrieve it throws a TimeZoneNotFoundException, the application can handle the exception in either of two ways:

- By calling the FromSerializedString method to deserialize a saved TimeZoneInfo object.
- By calling one of the overloads of the CreateCustomTimeZone method to create a time zone.

See the examples for the TimeZoneInfo.CreateCustomTimeZone(String, TimeSpan, String, String, String, TimeZoneInfo+AdjustmentRule[], Boolean) and FromSerializedString methods.

Constructors

Expand table

TimeZoneNotFoundException()	Initializes a new instance of the TimeZoneNotFoundException class with a system-supplied message.
TimeZoneNotFound Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TimeZoneNotFoundException class from serialized data.
TimeZoneNotFoundException(String)	Initializes a new instance of the TimeZoneNotFoundException class with the specified message string.
TimeZoneNotFoundException(String, Exception)	Initializes a new instance of the TimeZoneNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)

TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TimeZoneNotFoundException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TimeZoneNotFoundException class.

Overloads

Expand table

TimeZoneNotFoundException()	Initializes a new instance of the TimeZoneNotFoundException class with a system-supplied message.
TimeZoneNotFoundException(String)	Initializes a new instance of the TimeZoneNotFoundException class with the specified message string.
TimeZoneNotFoundException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TimeZoneNotFoundException class from serialized data.
TimeZoneNotFoundException(String, Exception)	Initializes a new instance of the TimeZoneNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

TimeZoneNotFoundException()

Initializes a new instance of the TimeZoneNotFoundException class with a system-supplied message.

```
C#
```

```
public TimeZoneNotFoundException ();
```

Remarks

This is the parameterless constructor of the TimeZoneNotFoundException class. This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "The time zone '*timeZoneName*' was not found on the local computer." This message is localized for the current system culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TimeZoneNotFoundException(String)

Initializes a new instance of the TimeZoneNotFoundException class with the specified message string.

C#

```
public TimeZoneNotFoundException (string? message);
```

Parameters

`message` String

A string that describes the exception.

Remarks

The `message` string is assigned to the Message property. The string should be localized for the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TimeZoneNotFoundException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the TimeZoneNotFoundException class from serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected TimeZoneNotFoundException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that contains the serialized data.

`context` StreamingContext

The stream that contains the serialized data.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` parameter is `null`.

-or-

The `context` parameter is `null`.

Remarks

This constructor is not called directly by your code to instantiate the TimeZoneNotFoundException object. Instead, it is called by the IFormatter object's Deserialize method when deserializing the TimeZoneNotFoundException object from a stream.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TimeZoneNotFoundException(String, Exception)

Initializes a new instance of the TimeZoneNotFoundException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public TimeZoneNotFoundException (string? message, Exception?
innerException);
```

Parameters

`message` String

A string that describes the exception.

`innerException` Exception

The exception that is the cause of the current exception.

Examples

The following example tries to retrieve a nonexistent time zone, which throws a TimeZoneNotFoundException. The exception handler wraps the exception in a new TimeZoneNotFoundException object, which the exception handler returns to the caller. The caller's exception handler then displays information about both the outer and inner exception.

C#

```
private void HandleInnerException()
{
   string timeZoneName = "Any Standard Time";
   TimeZoneInfo tz;
   try
   {
      tz = RetrieveTimeZone(timeZoneName);
      Console.WriteLine("The time zone display name is {0}.",
tz.DisplayName);
   }
   catch (TimeZoneNotFoundException e)
   {
      Console.WriteLine("{0} thrown by application", e.GetType().Name);
      Console.WriteLine("   Message: {0}", e.Message);
      if (e.InnerException != null)
      {
         Console.WriteLine("   Inner Exception Information:");
         Exception innerEx = e.InnerException;
         while (innerEx != null)
         {
            Console.WriteLine("      {0}: {1}", innerEx.GetType().Name,
innerEx.Message);
            innerEx = innerEx.InnerException;
         }
      }
   }
}

private TimeZoneInfo RetrieveTimeZone(string tzName)
{
   try
   {
```

```
        return TimeZoneInfo.FindSystemTimeZoneById(tzName);
    }
    catch (TimeZoneNotFoundException ex1)
    {
        throw new TimeZoneNotFoundException(
              String.Format("The time zone '{0}' cannot be found.",
tzName),
              ex1);
    }
    catch (InvalidTimeZoneException ex2)
    {
        throw new InvalidTimeZoneException(
              String.Format("The time zone {0} contains invalid data.",
tzName),
              ex2);
    }
}
```

Remarks

Typically, you use this TimeZoneNotFoundException overload to handle an exception in a `try`...`catch` block. The `innerException` parameter should be a reference to the exception object handled in the `catch` block, or it can be `null`. This value is then assigned to the TimeZoneNotFoundException object's InnerException property.

The `message` string is assigned to the Message property. The string should be localized for the current culture.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Tuple Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides static methods for creating tuple objects.

C#

```
public static class Tuple
```

Inheritance Object → Tuple

Examples

The following example creates an 8-tuple (octuple) that contains prime numbers that are less than 20.

C#

```
var primes = Tuple.Create(2, 3, 5, 7, 11, 13, 17, 19);
Console.WriteLine("Prime numbers less than 20: " +
                  "{0}, {1}, {2}, {3}, {4}, {5}, {6}, and {7}",
                  primes.Item1, primes.Item2, primes.Item3,
                  primes.Item4, primes.Item5, primes.Item6,
                  primes.Item7, primes.Rest.Item1);
// The example displays the following output:
//    Prime numbers less than 20: 2, 3, 5, 7, 11, 13, 17, and 19
```

Remarks

A tuple is a data structure that has a specific number and sequence of elements. An example of a tuple is a data structure with three elements (known as a 3-tuple or triple) that is used to store an identifier such as a person's name in the first element, a year in the second element, and the person's income for that year in the third element. The .NET Framework directly supports tuples with one to seven elements. In addition, you

can create tuples of eight or more elements by nesting tuple objects in the Rest property of a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Tuples are commonly used in four ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set.
- To return multiple values from a method without using `out` parameters (in C#) or `ByRef` parameters (in Visual Basic).
- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup time. If you supply a Tuple<T1,T2,T3> object as the method argument, you can supply the thread's startup routine with three items of data.

The Tuple class does not itself represent a tuple. Instead, it is a class that provides static methods for creating instances of the tuple types that are supported by the .NET Framework. It provides helper methods that you can call to instantiate tuple objects without having to explicitly specify the type of each tuple component.

Although you can create an instance of a tuple class by calling its class constructor, the code to do so can be cumbersome. The following example uses a class constructor to create a 7-tuple or septuple that contains population data for New York City for each census from 1950 through 2000.

C#

```
// Create a 7-tuple.
var population = new Tuple<string, int, int, int, int, int, int>(
                           "New York", 7891957, 7781984,
                           7894862, 7071639, 7322564, 8008278);
// Display the first and last elements.
Console.WriteLine("Population of {0} in 2000: {1:N0}",
                  population.Item1, population.Item7);
// The example displays the following output:
//       Population of New York in 2000: 8,008,278
```

Creating the same tuple object by using a helper method is more straightforward, as the following example shows.

C#

```
// Create a 7-tuple.
var population = Tuple.Create("New York", 7891957, 7781984, 7894862,
7071639, 7322564, 8008278);
// Display the first and last elements.
Console.WriteLine("Population of {0} in 2000: {1:N0}",
                  population.Item1, population.Item7);
// The example displays the following output:
//       Population of New York in 2000: 8,008,278
```

The Create helper methods directly support the creation of tuple objects that have from one to eight components (that is, singletons through octuples). Although there is no practical limit to the number of components a tuple may have, helper methods are not available to create a tuple with nine or more components. To create such a tuple, you must call the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> constructor.

> **Note**
>
> For additional information and examples that use tuples, see the documentation for the individual tuple types in the .NET Framework. These are listed in the See Also section at the end of this topic.

Methods

Expand table

Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Creates a new 8-tuple, or octuple.
Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Creates a new 7-tuple, or septuple.
Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Creates a new 6-tuple, or sextuple.
Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Creates a new 5-tuple, or quintuple.
Create<T1,T2,T3,T4>(T1, T2, T3, T4)	Creates a new 4-tuple, or quadruple.
Create<T1,T2,T3>(T1, T2, T3)	Creates a new 3-tuple, or triple.
Create<T1,T2>(T1, T2)	Creates a new 2-tuple, or pair.
Create<T1>(T1)	Creates a new 1-tuple, or singleton.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple.Create Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new tuple object.

Overloads

Expand table

Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Creates a new 8-tuple, or octuple.
Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Creates a new 7-tuple, or septuple.
Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Creates a new 6-tuple, or sextuple.
Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Creates a new 5-tuple, or quintuple.
Create<T1,T2,T3,T4>(T1, T2, T3, T4)	Creates a new 4-tuple, or quadruple.
Create<T1,T2,T3>(T1, T2, T3)	Creates a new 3-tuple, or triple.
Create<T1,T2>(T1, T2)	Creates a new 2-tuple, or pair.
Create<T1>(T1)	Creates a new 1-tuple, or singleton.

Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)

Creates a new 8-tuple, or octuple.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>
Create<T1,T2,T3,T4,T5,T6,T7,T8> (T1 item1, T2 item2, T3 item3, T4 item4,
```

```
  T5 item5, T6 item6, T7 item7, T8 item8);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

`T4`

The type of the fourth component of the tuple.

`T5`

The type of the fifth component of the tuple.

`T6`

The type of the sixth component of the tuple.

`T7`

The type of the seventh component of the tuple.

`T8`

The type of the eighth component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

`item4` T4

The value of the fourth component of the tuple.

`item5` T5

The value of the fifth component of the tuple.

`item6` T6

The value of the sixth component of the tuple.

`item7` T7

The value of the seventh component of the tuple.

`item8` T8

The value of the eighth component of the tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>

An 8-tuple (octuple) whose value is (`item1`, `item2`, `item3`, `item4`, `item5`, `item6`, `item7`, `item8`).

Examples

The following example creates an 8-tuple whose components are prime numbers that are less than 20.

C#

```
var primes = Tuple.Create(2, 3, 5, 7, 11, 13, 17, 19);
Console.WriteLine("Prime numbers less than 20: " +
                  "{0}, {1}, {2}, {3}, {4}, {5}, {6}, and {7}",
                  primes.Item1, primes.Item2, primes.Item3,
                  primes.Item4, primes.Item5, primes.Item6,
                  primes.Item7, primes.Rest.Item1);
// The example displays the following output:
//    Prime numbers less than 20: 2, 3, 5, 7, 11, 13, 17, and 19
```

This is equivalent to the following example, which uses the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class constructor instead of the Create factory creation method. Note that instantiating a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object in this way involves considerably more code, because you must declare a nested Tuple<T1> object as the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's eighth component to produce an octuple.

```C#
var primes = new Tuple<int, int, int, int, int, int, int,
                 Tuple<int>>(2, 3, 5, 7, 11, 13, 16,
                 new Tuple<int>(19));
```

Remarks

Create is a helper method that you can call to instantiate an 8-tuple without having to explicitly specify the types of its components.

> **Note**
>
> You must call the **Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>** constructor to create a tuple with nine or more components unless your language provides a special syntax for this purpose. The static (`Shared` in Visual Basic) methods of the **Tuple** class cannot be used to create a tuple with nine or more components.

See also

- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)

Creates a new 7-tuple, or septuple.

```
C#

public static Tuple<T1,T2,T3,T4,T5,T6,T7> Create<T1,T2,T3,T4,T5,T6,T7>
(T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6, T7 item7);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

`T4`

The type of the fourth component of the tuple.

`T5`

The type of the fifth component of the tuple.

`T6`

The type of the sixth component of the tuple.

`T7`

The type of the seventh component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

`item4` T4

The value of the fourth component of the tuple.

`item5` T5

The value of the fifth component of the tuple.

`item6` T6

The value of the sixth component of the tuple.

`item7` T7

The value of the seventh component of the tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7>

A 7-tuple whose value is (`item1`, `item2`, `item3`, `item4`, `item5`, `item6`, `item7`).

Remarks

Create is a helper method that you can call to instantiate a 7-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 7-tuple.

C#

```
var tuple7 = Tuple.Create("Jane", 90, 87, 93, 67, 100, 92);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}, {6}",
                  tuple7.Item1, tuple7.Item2, tuple7.Item3,
                  tuple7.Item4, tuple7.Item5, tuple7.Item6,
                  tuple7.Item7);
// Displays Test scores for Jane: 90, 87, 93, 67, 100, 92
```

This code is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5,T6,T7> class constructor.

C#

```
var tuple7 = new Tuple<string, int, int, int, int, int, int>
                      ("Jane", 90, 87, 93, 67, 100, 92);
```

```
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}, {6}",
                  tuple7.Item1, tuple7.Item2, tuple7.Item3,
                  tuple7.Item4, tuple7.Item5, tuple7.Item6,
                  tuple7.Item7);
// Displays Test scores for Jane: 90, 87, 93, 67, 100, 92
```

See also

- Tuple<T1,T2,T3,T4,T5,T6,T7>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)

Creates a new 6-tuple, or sextuple.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6> Create<T1,T2,T3,T4,T5,T6> (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6);
```

Type Parameters

T1

The type of the first component of the tuple.

T2

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

`T4`

The type of the fourth component of the tuple.

`T5`

The type of the fifth component of the tuple.

`T6`

The type of the sixth component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

`item4` T4

The value of the fourth component of the tuple.

`item5` T5

The value of the fifth component of the tuple.

`item6` T6

The value of the sixth component of the tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6>

A 6-tuple whose value is (`item1`, `item2`, `item3`, `item4`, `item5`, `item6`).

Remarks

Create is a helper method that you can call to instantiate a 6-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 6-tuple.

```
C#

var tuple6 = Tuple.Create("Jane", 90, 87, 93, 67, 100);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}",
                  tuple6.Item1, tuple6.Item2, tuple6.Item3,
                  tuple6.Item4, tuple6.Item5, tuple6.Item6);
// Displays Test scores for Jane: 90, 87, 93, 67, 100
```

This code is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5,T6> class constructor.

```
C#

var tuple6 = new Tuple<string, int, int, int, int, int>
                     ("Jane", 90, 87, 93, 67, 100);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}",
                  tuple6.Item1, tuple6.Item2, tuple6.Item3,
                  tuple6.Item4, tuple6.Item5, tuple6.Item6);
// Displays Test scores for Jane: 90, 87, 93, 67, 100
```

See also

- Tuple<T1,T2,T3,T4,T5,T6>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)

Creates a new 5-tuple, or quintuple.

```
C#

public static Tuple<T1,T2,T3,T4,T5> Create<T1,T2,T3,T4,T5> (T1 item1, T2
item2, T3 item3, T4 item4, T5 item5);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

`T4`

The type of the fourth component of the tuple.

`T5`

The type of the fifth component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

`item4` T4

The value of the fourth component of the tuple.

`item5` T5

The value of the fifth component of the tuple.

Returns

Tuple<T1,T2,T3,T4,T5>

A 5-tuple whose value is (`item1`, `item2`, `item3`, `item4`, `item5`).

Remarks

Create is a helper method that you can call to instantiate a 5-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 5-tuple.

```
C#

var tuple5 = Tuple.Create("New York", 1990, 7322564, 2000, 8008278);
Console.WriteLine("{0}: {1:N0} in {2}, {3:N0} in {4}",
                  tuple5.Item1, tuple5.Item3, tuple5.Item2,
                  tuple5.Item5, tuple5.Item4);
// Displays New York: 7,322,564 in 1990, 8,008,278 in 2000
```

This code is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5> class constructor.

```
C#

var tuple5 = new Tuple<string, int, int, int, int>
                      ("New York", 1990, 7322564, 2000, 8008278);
Console.WriteLine("{0}: {1:N0} in {2}, {3:N0} in {4}",
                  tuple5.Item1, tuple5.Item3, tuple5.Item2,
                  tuple5.Item5, tuple5.Item4);
// Displays New York: 7,322,564 in 1990, 8,008,278 in 2000
```

See also

- Tuple<T1,T2,T3,T4,T5>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2,T3,T4>(T1, T2, T3, T4)

Creates a new 4-tuple, or quadruple.

C#

```
public static Tuple<T1,T2,T3,T4> Create<T1,T2,T3,T4> (T1 item1, T2 item2,
T3 item3, T4 item4);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

`T4`

The type of the fourth component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

`item4` T4

The value of the fourth component of the tuple.

Returns

Tuple<T1,T2,T3,T4>

A 4-tuple whose value is (`item1`, `item2`, `item3`, `item4`).

Remarks

Create is a helper method that you can call to instantiate a 4-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 4-tuple.

C#

```
var tuple4 = Tuple.Create("New York", 32.68, 51.87, 76.3);
Console.WriteLine("{0}: Hi {1}, Lo {2}, Ave {3}",
                  tuple4.Item1, tuple4.Item4, tuple4.Item2,
                  tuple4.Item3);
// Displays New York: Hi 76.3, Lo 32.68, Ave 51.87
```

This code is equivalent to the following call to the Tuple<T1,T2,T3,T4>.Tuple<T1,T2,T3,T4> class constructor.

C#

```
var tuple4 = new Tuple<string, double, double, double>
                      ("New York", 32.68, 51.87, 76.3);
Console.WriteLine("{0}: Hi {1}, Lo {2}, Ave {3}",
                  tuple4.Item1, tuple4.Item4, tuple4.Item2,
                  tuple4.Item3);
// Displays New York: Hi 76.3, Lo 32.68, Ave 51.87
```

See also

- Tuple<T1,T2,T3,T4>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2,T3>(T1, T2, T3)

Creates a new 3-tuple, or triple.

C#

```
public static Tuple<T1,T2,T3> Create<T1,T2,T3> (T1 item1, T2 item2, T3
item3);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

`T3`

The type of the third component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

`item3` T3

The value of the third component of the tuple.

Returns

Tuple<T1,T2,T3>

A 3-tuple whose value is (`item1`, `item2`, `item3`).

Remarks

Create is a helper method that you can call to instantiate a 3-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 3-tuple.

```
C#

var tuple3 = Tuple.Create("New York", 32.68, 51.87);
Console.WriteLine("{0}: lo {1}, hi {2}",
                  tuple3.Item1, tuple3.Item2, tuple3.Item3);
// Displays New York: lo 32.68, hi 51.87
```

This code is equivalent to the following call to the Tuple<T1,T2,T3>.Tuple<T1,T2,T3> class constructor.

```
C#

var tuple3 = new Tuple<string, double, double>
                      ("New York", 32.68, 51.87);
Console.WriteLine("{0}: lo {1}, hi {2}",
                  tuple3.Item1, tuple3.Item2, tuple3.Item3);
// Displays New York: lo 32.68, hi 51.87
```

See also

- Tuple<T1,T2,T3>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1,T2>(T1, T2)

Creates a new 2-tuple, or pair.

```
C#

public static Tuple<T1,T2> Create<T1,T2> (T1 item1, T2 item2);
```

Type Parameters

`T1`

The type of the first component of the tuple.

`T2`

The type of the second component of the tuple.

Parameters

`item1` T1

The value of the first component of the tuple.

`item2` T2

The value of the second component of the tuple.

Returns

Tuple<T1,T2>

A 2-tuple whose value is (`item1`, `item2`).

Remarks

Create is a helper method that you can call to instantiate a 2-tuple object without having to explicitly specify the types of its components. The following example uses the Create method to instantiate a 2-tuple.

```C#
var tuple2 = Tuple.Create("New York", 32.68);
Console.WriteLine("{0}: {1}", tuple2.Item1, tuple2.Item2);
// Displays New York: 32.68
```

This code is equivalent to the following call to the Tuple<T1,T2> class constructor.

```C#
var tuple2 = new Tuple<string, double>("New York", 32.68);
Console.WriteLine("{0}: {1}", tuple2.Item1, tuple2.Item2);
// Displays New York: 32.68
```

See also

- Tuple<T1,T2>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Create<T1>(T1)

Creates a new 1-tuple, or singleton.

```C#
public static Tuple<T1> Create<T1> (T1 item1);
```

Type Parameters

T1

The type of the only component of the tuple.

Parameters

item1 T1

The value of the only component of the tuple.

Returns

Tuple<T1>

A tuple whose value is (`item1`).

Remarks

Create is a helper method that you can call to instantiate a 1-tuple object without having to explicitly specify the type of its component. The following example uses the Create method to instantiate a 1-tuple whose component is of type Int32.

```C#
var tuple1 = Tuple.Create(12);
Console.WriteLine(tuple1.Item1);     // Displays 12
```

This code is equivalent to the following call to the Tuple<T1> class constructor.

```C#
var tuple1 = new Tuple<int>(12);
Console.WriteLine(tuple1.Item1);     // Displays 12
```

See also

- Tuple<T1>

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 1-tuple, or singleton.

C#

```
public class Tuple<T1> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's only component.

Inheritance Object → Tuple<T1>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1> class represents a 1-tuple, or singleton, which is a tuple that has a single component. A singleton is used comparatively rarely in application development.

You can instantiate a Tuple<T1> object by calling either the Tuple<T1> constructor or the static Tuple.Create method. You can retrieve the value of the tuple's single component by using the read-only Item1 instance property.

Constructors

Expand table

Tuple<T1>(T1)	Initializes a new instance of the Tuple<T1> class.

Properties

Expand table

Item1	Gets the value of the Tuple<T1> object's single component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1> instance.

Explicit Interface Implementations

Expand table

IComparable.Compare To(Object)	Compares the current Tuple<T1> object to a specified object, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable. CompareTo(Object, IComparer)	Compares the current Tuple<T1> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable. Equals(Object, IEquality Comparer)	Returns a value that indicates whether the current Tuple<T1> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.Get HashCode(IEquality	Calculates the hash code for the current Tuple<T1> object by using a specified computation method.

Comparer)	
ITuple.Item[Int32]	Gets the value of the `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1>(Tuple<T1>, T1)	Deconstructs a tuple with 1 element into a separate variable.
ToValueTuple<T1>(Tuple<T1>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1>(T1) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1> class.

```
C#

public Tuple (T1 item1);
```

Parameters

item1 T1

The value of the tuple's only component.

Remarks

You can also use the static Tuple.Create method to instantiate a 1-tuple object without having to explicitly specify the type of its component. The following example uses the Create method to instantiate a 1-tuple whose component is of type Int32.

```
C#

var tuple1 = Tuple.Create(12);
Console.WriteLine(tuple1.Item1);     // Displays 12
```

This is equivalent to the following call to the Tuple<T1> class constructor.

```
C#

var tuple1 = new Tuple<int>(12);
Console.WriteLine(tuple1.Item1);     // Displays 12
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the Tuple<T1> object's single component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1> object's single component.

Examples

The following example displays information about two singletons and their components.

C#

```
using System;
using System.Numerics;

public class Example
{
   public static void Main()
   {
      var tuple1 = Tuple.Create(-1.23445e-32);
      // Display information about this singleton.
      Type tuple1Type = tuple1.GetType();
      Console.WriteLine("First 1-Tuple:");
      Console.WriteLine("   Type: {0}", tuple1Type.Name);
      Console.WriteLine("   Generic Parameter Type: {0}",
                        tuple1Type.GetGenericArguments()[0]);
      Console.WriteLine("   Component Value: {0}", tuple1.Item1);
      Console.WriteLine("   Component Value Type: {0}",
                        tuple1.Item1.GetType().Name);
      Console.WriteLine();
```

```
        var tuple2 =
 Tuple.Create((BigInteger)1.83789322281780983781356676e103);
        // Display information about this singleton.
        Type tuple2Type = tuple2.GetType();
        Console.WriteLine("Second 1-Tuple:");
        Console.WriteLine("   Type: {0}", tuple2Type.Name);
        Console.WriteLine("   Generic Parameter Type: {0}",
                          tuple2Type.GetGenericArguments()[0]);
        Console.WriteLine("   Component Value: {0}", tuple2.Item1);
        Console.WriteLine("   Component Value Type: {0}",
                          tuple2.Item1.GetType().Name);
    }
 }
 // The example displays the following output:
 //       First 1-Tuple:
 //          Type: Tuple`1
 //          Generic Parameter Type: System.Double
 //          Component Value: -1.23445E-32
 //          Component Value Type: Double
 //
 //       Second 1-Tuple:
 //          Type: Tuple`1
 //          Generic Parameter Type: System.Numerics.BigInteger
 //          Component Value: 1.83789322281780981688589094 92E+103
 //          Component Value Type: BigInteger
```

Remarks

You can determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

The example illustrates both approaches.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1> object is equal to a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example calls the Tuple<T1>.Equals(Object) method to compare a Tuple<T1> object whose component is a Double value with three Tuple<T1> objects whose components have the following characteristics:

- Same type (Double) and same value.
- Same type (Double), but different value.
- Different type (Single), but same value.

```
C#

using System;
```

```
public class Example
{
   public static void Main()
   {
      var doubleTuple1 = Tuple.Create(12.3455);
      var doubleTuple2 = Tuple.Create(16.8912);
      var doubleTuple3 = Tuple.Create(12.3455);
      var singleTuple1 = Tuple.Create(12.3455f);
      var tuple2 = Tuple.Create("James", 97.3);

      // Compare first tuple with a Tuple(Of Double) with a different value.
      TestEquality(doubleTuple1, doubleTuple2);
      // Compare first tuple with a Tuple(Of Double) with the same value.
      TestEquality(doubleTuple1, doubleTuple3);
      // Compare first tuple with a Tuple(Of Single) with the same value.
      TestEquality(doubleTuple1, singleTuple1);
      // Compare a 1-tuple with a 2-tuple.
      TestEquality(doubleTuple1, tuple2);
   }

   private static void TestEquality(Tuple<double> tuple, object obj)
   {
      Console.WriteLine("{0} = {1}: {2}", tuple.ToString(),
                                          obj.ToString(),
                                          tuple.Equals(obj));
   }
}
// The example displays the following output:
//       (12.3455) = (16.8912): False
//       (12.3455) = (12.3455): True
//       (12.3455) = (12.3455): False
//       (12.3455) = (James, 97.3): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1> object.
- Its single component is of the same type as the current instance.
- Its single component is equal to that of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Tuple<T1>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1> object.

Examples

The following example illustrates the ToString method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      var tuple1Double = Tuple.Create(3.456e-18);
      DisplayTuple(tuple1Double);

      var tuple1String = Tuple.Create("Australia");
      DisplayTuple(tuple1String);

      var tuple1Bool = Tuple.Create(true);
      DisplayTuple(tuple1Bool);

      var tuple1Char = Tuple.Create('a');
      DisplayTuple(tuple1Char);
   }
```

```
    private static void DisplayTuple(object obj)
    {
        Console.WriteLine(obj.ToString());
    }
}
// The example displays the following output:
//       (3.456E-18)
//       (Australia)
//       (True)
//       (a)
```

Remarks

The string returned by this method takes the form (*Item1*), where *Item1* represents the value of the Item1 property. If the value of Item1 is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1>.IStructuralComparable.CompareTo(Object, IComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

An object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1> object.

Examples

The following example defines a generic class named `DescendingComparer` that implements the IComparer<T> interface. `DescendingComparer` sorts objects in descending rather than ascending order by reversing the value returned by the default comparer for a particular type. An instance of the generic `DescendingComparer` class is then passed to the Array.Sort(Array, IComparer) method to sort an array of Tuple<T1> objects in descending order. Note that the example does not directly call the IStructuralComparable.CompareTo method. This method is called implicitly by the Array.Sort(Array, IComparer) method for each element in the array.

C#

```
using System;
using System.Collections.Generic;

public class DescendingComparer<T> : IComparer<T>
{
    public int Compare(T x, T y)
    {
        return -1 * Comparer<T>.Default.Compare(x, y);
    }
}

class CompareTo2
{
   static void Main()
   {
       Tuple<Double>[] values = { Tuple.Create(13.54),
                                  Tuple.Create(Double.NaN),
                                  Tuple.Create(-189.42993),
                                  Tuple.Create(Double.PositiveInfinity),
                                  Tuple.Create(Double.Epsilon),
```

```
                                    Tuple.Create(1.934E-17),
                                    Tuple.Create(Double.NegativeInfinity),
                                    Tuple.Create(-0.000000000003588),
                                    null };
        Console.WriteLine("The values in unsorted order:");
        foreach (var value in values)
            if (value != null)
                Console.WriteLine("   {0}", value.Item1);
            else
                Console.WriteLine("   <null>");
        Console.WriteLine();

        Array.Sort(values, new DescendingComparer<Tuple<Double>>());

        Console.WriteLine("The values sorted in descending order:");
        foreach (var value in values)
            if (value != null)
                Console.WriteLine("   {0}", value.Item1);
            else
                Console.WriteLine("   <null>");
    }
}
// The example displays the following output:
//      The values in unsorted order:
//         13.54
//         NaN
//         -189.42993
//         Infinity
//         4.94065645841247E-324
//         1.934E-17
//         -Infinity
//         -3.588E-12
//
//      The values sorted in descending order:
//         Infinity
//         13.54
//         1.934E-17
//         4.94065645841247E-324
//         -3.588E-12
//         -189.42993
//         -Infinity
//         NaN
```

Remarks

Although this method can be called directly, it is most commonly called by collection sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

⊗ **Caution**

The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IStructuralEquatable.Equals(Object, IEqualityComparer)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1>.IStructuralEquatable.Equals(Object, IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines an IEqualityComparer implementation that considers two floating-point values to be equal if they are approximately equal to each other (that is, if one value is within .01 percent of the other).

C#

```
using System;
using System.Collections;

public class Tuple1Comparer : IEqualityComparer
{
   new public bool Equals(object x, object y)
   {
      // Check if x is a floating point type. If x is, then y is.
      if (x is double | x is float)
      {
         // Convert to Double values.
         double dblX = (double) x;
         double dblY = (double) y;
         if (Double.IsNaN(dblX) | Double.IsInfinity(dblX) |
             Double.IsNaN(dblY) | Double.IsInfinity(dblY))
            return dblX.Equals(dblY);
         else
            return Math.Abs(dblX - dblY) <= dblX * .0001;
      }
      else
      {
         return x.Equals(y);
      }
   }

   public int GetHashCode(object obj)
   {
      return obj.GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      var doubleTuple1 = Tuple.Create(12.3455);

      var doubleTuple2 = Tuple.Create(16.8912);
      var doubleTuple3 = Tuple.Create(12.3449599);

      // Compare first tuple with a Tuple<double> with a different value.
      TestEquality(doubleTuple1, doubleTuple2);
      //Compare first tuple with a Tuple<double> with the same value.
      TestEquality(doubleTuple1, doubleTuple3);
   }

   private static void TestEquality(Tuple<double> tuple, object obj)
```

```
    {
       Console.WriteLine("{0} = {1}: {2}", tuple.ToString(),
                                           obj.ToString(),

((IStructuralEquatable)tuple).Equals(obj, new Tuple1Comparer()));
    }
}
// The example displays the following output:
//       (12.3455) = (16.8912): False
//       (12.3455) = (12.3449599): True
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1> object whose single component is of the same type as the current instance. The method is passed the Item1 component of the current instance and the Item1 component of the Tuple<T1> object represented by the `other` parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IEqualityComparer

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Tuple<T1>.IStructuralEquatable.GetHashCode(IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

comparer IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1> object to a specified object, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1> object.

Examples

The following example creates an array of singletons whose component is a Double value. It displays the value of each tuple component in unsorted order, sorts the array, and then displays the values in sorted order. Note that the example does not directly call the Tuple<T1>.IComparable.CompareTo method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

class Example
{
   static void Main()
   {
      Tuple<Double>[] values = { Tuple.Create(13.54),
                                 Tuple.Create(Double.NaN),
                                 Tuple.Create(-189.42993),
                                 Tuple.Create(Double.PositiveInfinity),
                                 Tuple.Create(Double.Epsilon),
                                 Tuple.Create(1.934E-17),
                                 Tuple.Create(Double.NegativeInfinity),
                                 Tuple.Create(-0.000000000003588),
                                 null };
      Console.WriteLine("The values in unsorted order:");
      foreach (var value in values)
         if (value != null)
            Console.WriteLine("   {0}", value.Item1);
         else
            Console.WriteLine("   <null>");
      Console.WriteLine();

      Array.Sort(values);

      Console.WriteLine("The values in sorted order:");
      foreach (var value in values)
         if (value != null)
            Console.WriteLine("   {0}", value.Item1);
         else
            Console.WriteLine("   <null>");
   }
```

```
}
// The example displays the following output:
//      The values in unsorted order:
//         13.54
//         NaN
//         -189.42993
//         Infinity
//         4.94065645841247E-324
//         1.934E-17
//         -Infinity
//         -3.588E-12
//
//      The values in sorted order:
//         NaN
//         -Infinity
//         -189.42993
//         -3.588E-12
//         4.94065645841247E-324
//         1.934E-17
//         13.54
//         Infinity
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1> class. Although the method can be called directly, it is most commonly called by the default overloads of collection sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **Tuple<T1>.IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Equals** method.

The Tuple<T1>.IComparable.CompareTo method uses the default object comparer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the `Tuple` element. `index` must be 0.

Property Value

Object

The value of the `Tuple` element.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 0.

Remarks

The Tuple<T1>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

1, the number of elements in a Tuple<T1> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 2-tuple, or pair.

C#

```
public class Tuple<T1,T2> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

Inheritance Object → Tuple<T1,T2>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2> class represents a 2-tuple, or pair, which is a tuple that has two components. A 2-tuple is similar to a KeyValuePair<TKey,TValue> structure.

You can instantiate a Tuple<T1,T2> object by calling either the Tuple<T1,T2> constructor or the static Tuple.Create<T1,T2>(T1, T2) method. You can retrieve the values of the tuple's components by using the read-only Item1 and Item2 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a record in a database, and its components can represent that record's fields.

- To provide easy access to, and manipulation of, a data set. The following example defines an array of Tuple<T1,T2> objects that contain the names of students and their corresponding test scores. It then iterates the array to calculate the mean test score.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, Nullable<int>>[] scores =
                    { new Tuple<string, Nullable<int>>("Jack", 78),
                      new Tuple<string, Nullable<int>>("Abbey", 92),
                      new Tuple<string, Nullable<int>>("Dave", 88),
                      new Tuple<string, Nullable<int>>("Sam", 91),
                      new Tuple<string, Nullable<int>>("Ed", null),
                      new Tuple<string, Nullable<int>>("Penelope", 82),
                      new Tuple<string, Nullable<int>>("Linda", 99),
                      new Tuple<string, Nullable<int>>("Judith", 84) };
      int number;
      double mean = ComputeMean(scores, out number);
      Console.WriteLine("Average test score: {0:N2} (n={1})", mean,
number);
   }

   private static double ComputeMean(Tuple<string, Nullable<int>>[]
scores, out int n)
   {
      n = 0;
      int sum = 0;
      foreach (var score in scores)
      {
         if (score.Item2.HasValue)
         {
            n += 1;
            sum += score.Item2.Value;
         }
      }
      if (n > 0)
         return sum / (double) n;
      else
         return 0;
   }
}
```

```
// The example displays the following output:
//       Average test score: 87.71 (n=7)
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the following example uses a Tuple<T1,T2> object to return the quotient and the remainder that result from integer division.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      int dividend, divisor;
      Tuple<int, int> result;

      dividend = 136945; divisor = 178;
      result = IntegerDivide(dividend, divisor);
      if (result != null)
         Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                           dividend, divisor, result.Item1,
result.Item2);
      else
         Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);

      dividend = Int32.MaxValue; divisor = -2073;
      result = IntegerDivide(dividend, divisor);
      if (result != null)
         Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                           dividend, divisor, result.Item1,
result.Item2);
      else
         Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);
   }

   private static Tuple<int, int> IntegerDivide(int dividend, int
divisor)
   {
      try {
         int remainder;
         int quotient = Math.DivRem(dividend, divisor, out remainder);
         return new Tuple<int, int>(quotient, remainder);
      }
      catch (DivideByZeroException) {
         return null;
      }
   }
}
// The example displays the following output:
```

```
//        136945 \ 178 = 769, remainder 63
//        2147483647 \ -2073 = -1035930, remainder 757
```

- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2> object as the method argument, you can supply the thread's startup routine with two items of data.

Constructors

Expand table

Tuple<T1,T2>(T1, T2)	Initializes a new instance of the Tuple<T1,T2> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2> object's first component.
Item2	Gets the value of the current Tuple<T1,T2> object's second component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current Tuple<T1,T2> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current Tuple<T1,T2> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current Tuple<T1,T2> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2>(Tuple<T1,T2>, T1, T2)	Deconstructs a tuple with 2 elements into separate variables.
ToValueTuple<T1,T2>(Tuple<T1,T2>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>(T1, T2) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2> class.

```
C#

public Tuple (T1 item1, T2 item2);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

Remarks

You can also use the static Tuple.Create<T1,T2>(T1, T2) method to instantiate a 2-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2>(T1, T2) method to instantiate a 2-tuple whose components are of type String and Double.

```
C#

var tuple2 = Tuple.Create("New York", 32.68);
Console.WriteLine("{0}: {1}", tuple2.Item1, tuple2.Item2);
// Displays New York: 32.68
```

This is equivalent to the following call to the Tuple<T1,T2> class constructor.

```
C#

var tuple2 = new Tuple<string, double>("New York", 32.68);
Console.WriteLine("{0}: {1}", tuple2.Item1, tuple2.Item2);
```

```
// Displays New York: 32.68
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2> object's first component.

Examples

The example illustrates the use of the Item1 and Item2 properties to define a method that returns multiple values in the form of a 2-tuple.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      int dividend, divisor;
      Tuple<int, int> result;

      dividend = 136945; divisor = 178;
      result = IntegerDivide(dividend, divisor);
      if (result != null)
         Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                           dividend, divisor, result.Item1, result.Item2);
      else
         Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);
```

```
        dividend = Int32.MaxValue; divisor = -2073;
        result = IntegerDivide(dividend, divisor);
        if (result != null)
           Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                             dividend, divisor, result.Item1, result.Item2);
        else
           Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);
     }

     private static Tuple<int, int> IntegerDivide(int dividend, int divisor)
     {
        try {
           int remainder;
           int quotient = Math.DivRem(dividend, divisor, out remainder);
           return new Tuple<int, int>(quotient, remainder);
        }
        catch (DivideByZeroException) {
           return null;
        }
     }
  }
  // The example displays the following output:
  //       136945 \ 178 = 769, remainder 63
  //       2147483647 \ -2073 = -1035930, remainder 757
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2> object's second component.

Examples

The example illustrates the use of the Item1 and Item2 properties to define a method that returns multiple values in the form of a 2-tuple.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      int dividend, divisor;
      Tuple<int, int> result;

      dividend = 136945; divisor = 178;
      result = IntegerDivide(dividend, divisor);
      if (result != null)
         Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                           dividend, divisor, result.Item1, result.Item2);
      else
         Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);
```

```
        dividend = Int32.MaxValue; divisor = -2073;
        result = IntegerDivide(dividend, divisor);
        if (result != null)
           Console.WriteLine(@"{0} \ {1} = {2}, remainder {3}",
                             dividend, divisor, result.Item1, result.Item2);
        else
           Console.WriteLine(@"{0} \ {1} = <Error>", dividend, divisor);
     }

     private static Tuple<int, int> IntegerDivide(int dividend, int divisor)
     {
        try {
           int remainder;
           int quotient = Math.DivRem(dividend, divisor, out remainder);
           return new Tuple<int, int>(quotient, remainder);
        }
        catch (DivideByZeroException) {
           return null;
        }
     }
  }
  // The example displays the following output:
  //       136945 \ 178 = 769, remainder 63
  //       2147483647 \ -2073 = -1035930, remainder 757
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2> object is equal to a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example calls the Tuple<T1,T2>.Equals(Object) method to determine whether any of the objects in an array of Tuple<T1,T2> objects are equal to one another. The output reflects the fact that the Equals(Object) method returns `true` when comparing Tuple<T1,T2> objects whose components have equal values.

```
C#

using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, Nullable<int>>[] scores =
```

```
                        { new Tuple<string, Nullable<int>>("Dan", 90),
                          new Tuple<string, Nullable<int>>("Ernie", null),
                          new Tuple<string, Nullable<int>>("Jill", 88),
                          new Tuple<string, Nullable<int>>("Ernie", null),
                          new Tuple<string, Nullable<int>>("Nancy", 88),
                          new Tuple<string, Nullable<int>>("Dan", 90) };

      // Compare the Tuple objects
      for (int ctr = 0; ctr < scores.Length; ctr++)
      {
         for (int innerCtr = ctr + 1; innerCtr < scores.Length; innerCtr++)
         {
            Console.WriteLine("{0} = {1}: {2}",
                              scores[ctr], scores[innerCtr],
                              scores[ctr].Equals(scores[innerCtr]));
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       (Dan, 90) = (Ernie, ): False
//       (Dan, 90) = (Jill, 88): False
//       (Dan, 90) = (Ernie, ): False
//       (Dan, 90) = (Nancy, 88): False
//       (Dan, 90) = (Dan, 90): True
//
//       (Ernie, ) = (Jill, 88): False
//       (Ernie, ) = (Ernie, ): True
//       (Ernie, ) = (Nancy, 88): False
//       (Ernie, ) = (Dan, 90): False
//
//       (Jill, 88) = (Ernie, ): False
//       (Jill, 88) = (Nancy, 88): False
//       (Jill, 88) = (Dan, 90): False
//
//       (Ernie, ) = (Nancy, 88): False
//       (Ernie, ) = (Dan, 90): False
//
//       (Nancy, 88) = (Dan, 90): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2> object.
- Its two components are of the same types as the current instance.

- Its two components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Tuple<T1,T2>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2> object.

Examples

The following example illustrates the ToString method.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      Tuple<string, Nullable<int>>[] scores =
                      { new Tuple<string, Nullable<int>>("Abbey", 92),
                        new Tuple<string, Nullable<int>>("Dave", 88),
                        new Tuple<string, Nullable<int>>("Ed", null),
                        new Tuple<string, Nullable<int>>("Jack", 78),
                        new Tuple<string, Nullable<int>>("Linda", 99),
                        new Tuple<string, Nullable<int>>("Judith", 84),
                        new Tuple<string, Nullable<int>>("Penelope", 82),
                        new Tuple<string, Nullable<int>>("Sam", 91) };
      foreach (var score in scores)
         Console.WriteLine(score.ToString());
   }
}
```

```
// The example displays the following output:
//       (Abbey, 92)
//       (Dave, 88)
//       (Ed, )
//       (Jack, 78)
//       (Linda, 99)
//       (Judith, 84)
//       (Penelope, 82)
//       (Sam, 91)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*), where *Item1* and *Item2* represent the values of the Item1 and Item2 properties. If either property value is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.IStructuralComparable.CompareTo(Object, IComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

An object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2> object.

Examples

The following example creates an array of Tuple<T1,T2> objects that consist of a student's name and test score. It displays the component of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display the value of each tuple in sorted order. To sort the array, the example defines a generic `ScoreComparer` class that implements the IComparer interface and sorts the Tuple<T1,T2> objects in ascending order by the value of their second component rather than their first component. Note that the example does not directly call the IStructuralComparable.CompareTo method. This method is called implicitly by the Array.Sort(Array, IComparer) method for each element in the array.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class ScoreComparer<T1, T2> : IComparer
{
   public int Compare(object x, object y)
   {
      Tuple<T1, T2> tX = x as Tuple<T1,T2>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
         Tuple<T1, T2> tY = y as Tuple<T1, T2>;
         return Comparer<T2>.Default.Compare(tX.Item2, tY.Item2);
      }
   }
```

```
}

public class Example
{
   public static void Main()
   {
      Tuple<string, Nullable<int>>[] scores =
                    { new Tuple<string, Nullable<int>>("Jack", 78),
                      new Tuple<string, Nullable<int>>("Abbey", 92),
                      new Tuple<string, Nullable<int>>("Dave", 88),
                      new Tuple<string, Nullable<int>>("Sam", 91),
                      new Tuple<string, Nullable<int>>("Ed", null),
                      new Tuple<string, Nullable<int>>("Penelope", 82),
                      new Tuple<string, Nullable<int>>("Linda", 99),
                      new Tuple<string, Nullable<int>>("Judith", 84) };

      Console.WriteLine("The values in unsorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());

      Console.WriteLine();

      Array.Sort(scores, new ScoreComparer<string, Nullable<int>>());

      Console.WriteLine("The values in sorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());
   }
}
// The example displays the following output;
//       The values in unsorted order:
//       (Jack, 78)
//       (Abbey, 92)
//       (Dave, 88)
//       (Sam, 91)
//       (Ed, )
//       (Penelope, 82)
//       (Linda, 99)
//       (Judith, 84)
//
//       The values in sorted order:
//       (Ed, )
//       (Jack, 78)
//       (Penelope, 82)
//       (Judith, 84)
//       (Dave, 88)
//       (Sam, 91)
//       (Abbey, 92)
//       (Linda, 99)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2> instance is cast to an IStructuralComparable interface.

Although this method can be called directly, it is most commonly called by collection sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IStructuralEquatable.Equals(Object, IEqualityComparer)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Tuple<T1,T2>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines an `Item2Comparer` class that implements the IEqualityComparer interface and changes the way in which Tuple<T1,T2> objects are

evaluated for equality. The method always returns `true` when it is passed the Item1 property values of two Tuple<T1,T2> objects, and it calls the IStructuralEquatable.Equals method to evaluate their Item2 property values. As a result, the method tests for equality based only on the value of the Item2 property. The output illustrates the result for a data set of Tuple<T1,T2> objects that record the names of runners and the distances that they ran.

C#

```
using System;
using System.Collections;

public class Item2Comparer<T1, T2> : IEqualityComparer
{
   new public bool Equals(object x, object y)
   {
      // Return true for all values of Item1.
      if (x is T1)
      //if (typeof(x) is string)
         return true;
      else
         return x.Equals(y);
   }

   public int GetHashCode(object obj)
   {
      if (obj is T1)
         return ((T1) obj).GetHashCode();
      else
         return ((T2) obj).GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      Tuple<string, double>[] distancesWalked = {
                        Tuple.Create("Jan", Double.NaN),
                        Tuple.Create("Joe", Double.NaN),
                        Tuple.Create("Adam", 1.36),
                        Tuple.Create("Selena", 2.01),
                        Tuple.Create("Jake", 1.36) };
      for (int ctr = 0; ctr < distancesWalked.Length; ctr++)
      {
         Tuple<string, double> distanceWalked = distancesWalked[ctr];
         for (int ctr2 = ctr + 1; ctr2 < distancesWalked.Length; ctr2++)
         {
            Console.WriteLine("{0} = {1}: {2}", distanceWalked,
                              distancesWalked[ctr2],

((IStructuralEquatable)distanceWalked).Equals(distancesWalked[ctr2],
```

```
                                                        new
Item2Comparer<string, double>()));
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//       (Jan, NaN) = (Joe, NaN): True
//       (Jan, NaN) = (Adam, 1.36): False
//       (Jan, NaN) = (Selena, 2.01): False
//       (Jan, NaN) = (Jake, 1.36): False
//
//       (Joe, NaN) = (Adam, 1.36): False
//       (Joe, NaN) = (Selena, 2.01): False
//       (Joe, NaN) = (Jake, 1.36): False
//
//       (Adam, 1.36) = (Selena, 2.01): False
//       (Adam, 1.36) = (Jake, 1.36): True
//
//       (Selena, 2.01) = (Jake, 1.36): False
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2> instance is cast to an IStructuralEquatable interface.

The IStructuralEquatable.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2> object whose components are of the same types as the current instance. The IStructuralEquatable.Equals method first passes the Item1 values of the Tuple<T1,T2> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2> objects.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IEqualityComparer

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.IStructuralEquatable.GetHashCode(IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2> object.

Examples

The following example creates an array of Tuple<T1,T2> objects that consist of a student's name and test score. It displays the component of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display the value of each tuple in sorted order. The output shows that the array has been sorted by its first component. Note that the example does not directly call the IComparable.CompareTo(Object) method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, Nullable<int>>[] scores =
                    { new Tuple<string, Nullable<int>>("Jack", 78),
                      new Tuple<string, Nullable<int>>("Abbey", 92),
                      new Tuple<string, Nullable<int>>("Dave", 88),
                      new Tuple<string, Nullable<int>>("Sam", 91),
                      new Tuple<string, Nullable<int>>("Ed", null),
                      new Tuple<string, Nullable<int>>("Penelope", 82),
                      new Tuple<string, Nullable<int>>("Linda", 99),
                      new Tuple<string, Nullable<int>>("Judith", 84) };

      Console.WriteLine("The values in unsorted order:");
      foreach (Tuple<string, Nullable<int>> score in scores)
         Console.WriteLine(score.ToString());

      Console.WriteLine();

      Array.Sort(scores);

      Console.WriteLine("The values in sorted order:");
      foreach (Tuple<string, Nullable<int>> score in scores)
         Console.WriteLine(score.ToString());
   }
}
```

```
// The example displays the following output;
//       The values in unsorted order:
//       (Jack, 78)
//       (Abbey, 92)
//       (Dave, 88)
//       (Sam, 91)
//       (Ed, )
//       (Penelope, 82)
//       (Linda, 99)
//       (Judith, 84)
//
//       The values in sorted order:
//       (Abbey, 92)
//       (Dave, 88)
//       (Ed, )
//       (Jack, 78)
//       (Judith, 84)
//       (Linda, 99)
//       (Penelope, 82)
//       (Sam, 91)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2> class. Although the method can be called directly, it is most commonly called by the default overloads of collection sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **Tuple<T1,T2>.IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Equals** method.

The Tuple<T1,T2>.IComparable.CompareTo method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 1.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 1.

Remarks

The Tuple<T1,T2>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

C#

```
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

2, the number of elements in a Tuple<T1,T2> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 3-tuple, or triple.

C#

```
public class Tuple<T1,T2,T3> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

Inheritance Object → Tuple<T1,T2,T3>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3> class represents a 3-tuple, or triple, which is a tuple that has three components.

You can instantiate a Tuple<T1,T2,T3> object by calling either the Tuple<T1,T2,T3> constructor or the static Tuple.Create<T1,T2,T3>(T1, T2, T3) method. You can retrieve

the values of the tuple's components by using the read-only Item1, Item2, and Item3 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set. The following example defines an array of Tuple<T1,T2,T3> objects that contain the names of students, their average test scores, and the number of tests taken. The array is passed to the `ComputeStatistics` method, which calculates the mean and standard deviation of the test scores.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
                      Tuple.Create("Penelope", 82.9, 8),
                      Tuple.Create("Linda", 99.0, 9),
                      Tuple.Create("Judith", 84.3, 9) };
      var result = ComputeStatistics(scores);
      Console.WriteLine("Mean score: {0:N2} (SD={1:N2}) (n={2})",
                        result.Item2, result.Item3, result.Item1);
   }

   private static Tuple<int, double, double>
ComputeStatistics(Tuple<string, double, int>[] scores)
   {
      int n = 0;
      double sum = 0;

      // Compute the mean.
      foreach (var score in scores)
      {
         n += score.Item3;
         sum += score.Item2 * score.Item3;
      }
      double mean = sum / n;

      // Compute the standard deviation.
```

```
        double ss = 0;
        foreach (var score in scores)
        {
            ss = Math.Pow(score.Item2 - mean, 2);
        }
        double sd = Math.Sqrt(ss/scores.Length);
        return Tuple.Create(scores.Length, mean, sd);
    }
}
// The example displays the following output:
//       Mean score: 87.02 (SD=0.96) (n=8)
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its summary test score statistics in a Tuple<T1,T2,T3> object.
- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2,T3> object as the method argument, you can supply the thread's startup routine with three items of data.

Constructors

Expand table

Tuple<T1,T2,T3>(T1, T2, T3)	Initializes a new instance of the Tuple<T1,T2,T3> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3> object's third component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2,T3> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3> instance.

Explicit Interface Implementations

Expand table

IComparable.Compare To(Object)	Compares the current Tuple<T1,T2,T3> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable. CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable. Equals(Object, IEquality Comparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.Get HashCode(IEquality Comparer)	Calculates the hash code for the current Tuple<T1,T2,T3> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2,T3>(Tuple<T1,T2,T3>, T1, T2, T3)	Deconstructs a tuple with 3 elements into separate variables.

ToValueTuple<T1,T2,T3> (Tuple<T1,T2,T3>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>(T1, T2, T3) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

Remarks

You can also use the static Tuple.Create<T1,T2,T3>(T1, T2, T3) method to instantiate a 3-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2,T3>(T1, T2, T3) method to instantiate a 3-tuple whose components are of type String, Double, and Double.

C#

```
var tuple3 = Tuple.Create("New York", 32.68, 51.87);
Console.WriteLine("{0}: lo {1}, hi {2}",
                  tuple3.Item1, tuple3.Item2, tuple3.Item3);
// Displays New York: lo 32.68, hi 51.87
```

This is equivalent to the following call to the Tuple<T1,T2,T3> class constructor.

C#

```
var tuple3 = new Tuple<string, double, double>
                      ("New York", 32.68, 51.87);
Console.WriteLine("{0}: lo {1}, hi {2}",
                  tuple3.Item1, tuple3.Item2, tuple3.Item3);
// Displays New York: lo 32.68, hi 51.87
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3> object's first component.

Examples

The following example defines an array of Tuple<T1,T2,T3> objects that contain the names of students, their average test scores, and the number of tests taken. The array is passed to the `ComputeStatistics` method, which calculates the mean score, standard deviation, and number of cases from which the statistics are calculated. These values are stored in the Tuple<T1,T2,T3> object that is returned by the `ComputeStatistics` method. The Item1 property contains the number of cases.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
```

```
                        Tuple.Create("Penelope", 82.9, 8),
                        Tuple.Create("Linda", 99.0, 9),
                        Tuple.Create("Judith", 84.3, 9) };
      var result = ComputeStatistics(scores);
      Console.WriteLine("Mean score: {0:N2} (SD={1:N2}) (n={2})",
                          result.Item2, result.Item3, result.Item1);
   }

   private static Tuple<int, double, double> ComputeStatistics(Tuple<string,
double, int>[] scores)
   {
      int n = 0;
      double sum = 0;

      // Compute the mean.
      foreach (var score in scores)
      {
         n += score.Item3;
         sum += score.Item2 * score.Item3;
      }
      double mean = sum / n;

      // Compute the standard deviation.
      double ss = 0;
      foreach (var score in scores)
      {
         ss = Math.Pow(score.Item2 - mean, 2);
      }
      double sd = Math.Sqrt(ss/scores.Length);
      return Tuple.Create(scores.Length, mean, sd);
   }
}
// The example displays the following output:
//       Mean score: 87.02 (SD=0.96) (n=8)
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3> object's second component.

Examples

The following example defines an array of Tuple<T1,T2,T3> objects that contain the names of students, their average test scores, and the number of tests taken. The array is passed to the `ComputeStatistics` method, which calculates the mean score, standard deviation, and number of cases from which the statistics are calculated. These values are stored in the Tuple<T1,T2,T3> object that is returned by the `ComputeStatistics` method. The Item2 property contains the mean test score.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
```

```
                        Tuple.Create("Penelope", 82.9, 8),
                        Tuple.Create("Linda", 99.0, 9),
                        Tuple.Create("Judith", 84.3, 9) };
      var result = ComputeStatistics(scores);
      Console.WriteLine("Mean score: {0:N2} (SD={1:N2}) (n={2})",
                          result.Item2, result.Item3, result.Item1);
   }

   private static Tuple<int, double, double> ComputeStatistics(Tuple<string,
double, int>[] scores)
   {
      int n = 0;
      double sum = 0;

      // Compute the mean.
      foreach (var score in scores)
      {
         n += score.Item3;
         sum += score.Item2 * score.Item3;
      }
      double mean = sum / n;

      // Compute the standard deviation.
      double ss = 0;
      foreach (var score in scores)
      {
         ss = Math.Pow(score.Item2 - mean, 2);
      }
      double sd = Math.Sqrt(ss/scores.Length);
      return Tuple.Create(scores.Length, mean, sd);
   }
}
// The example displays the following output:
//       Mean score: 87.02 (SD=0.96) (n=8)
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3> object's third component.

Examples

The following example defines an array of Tuple<T1,T2,T3> objects that contain the names of students, their average test scores, and the number of tests taken. The array is passed to the `ComputeStatistics` method, which calculates the mean score, standard deviation, and number of cases from which the statistics are calculated. These values are stored in the Tuple<T1,T2,T3> object that is returned by the `ComputeStatistics` method. The Item3 property contains the standard deviation.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
```

```
                        Tuple.Create("Penelope", 82.9, 8),
                        Tuple.Create("Linda", 99.0, 9),
                        Tuple.Create("Judith", 84.3, 9) };
      var result = ComputeStatistics(scores);
      Console.WriteLine("Mean score: {0:N2} (SD={1:N2}) (n={2})",
                          result.Item2, result.Item3, result.Item1);
   }

   private static Tuple<int, double, double> ComputeStatistics(Tuple<string,
double, int>[] scores)
   {
      int n = 0;
      double sum = 0;

      // Compute the mean.
      foreach (var score in scores)
      {
         n += score.Item3;
         sum += score.Item2 * score.Item3;
      }
      double mean = sum / n;

      // Compute the standard deviation.
      double ss = 0;
      foreach (var score in scores)
      {
         ss = Math.Pow(score.Item2 - mean, 2);
      }
      double sd = Math.Sqrt(ss/scores.Length);
      return Tuple.Create(scores.Length, mean, sd);
   }
}
// The example displays the following output:
//       Mean score: 87.02 (SD=0.96) (n=8)
```

Remarks

You can dynamically determine the type of the Item3 component in one of two ways:

- By calling the `GetType` method on the value returned by the Item3 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3> object, and retrieving the third element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3> object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example calls the Tuple<T1,T2,T3>.Equals(Object) method to determine whether any of the objects in an array of Tuple<T1,T2,T3> objects are equal to one another. The output reflects the fact that the Equals(Object) method returns `true` when comparing Tuple<T1,T2,T3> objects whose components have equal values.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
```

```
                        { Tuple.Create("Ed", 78.8, 8),
                          Tuple.Create("Abbey", 92.1, 9),
                          Tuple.Create("Ed", 71.2, 9),
                          Tuple.Create("Sam", 91.7, 8),
                          Tuple.Create("Ed", 71.2, 5),
                          Tuple.Create("Penelope", 82.9, 8),
                          Tuple.Create("Ed", 71.2, 9),
                          Tuple.Create("Judith", 84.3, 9) };

      // Test each tuple object for equality with every other tuple.
      for (int ctr = 0; ctr < scores.Length; ctr++)
      {
         var currentTuple = scores[ctr];
         for (int ctr2 = ctr + 1; ctr2 < scores.Length; ctr2++)
            Console.WriteLine("{0} = {1}: {2}", currentTuple, scores[ctr2],

currentTuple.Equals(scores[ctr2]));

         Console.WriteLine();
      }
   }
}
// The example displays the following output;
//    (Ed, 78.8, 8) = (Abbey, 92.1, 9): False
//    (Ed, 78.8, 8) = (Ed, 71.2, 9): False
//    (Ed, 78.8, 8) = (Sam, 91.7, 8): False
//    (Ed, 78.8, 8) = (Ed, 71.2, 5): False
//    (Ed, 78.8, 8) = (Penelope, 82.9, 8): False
//    (Ed, 78.8, 8) = (Ed, 71.2, 9): False
//    (Ed, 78.8, 8) = (Judith, 84.3, 9): False
//
//    (Abbey, 92.1, 9) = (Ed, 71.2, 9): False
//    (Abbey, 92.1, 9) = (Sam, 91.7, 8): False
//    (Abbey, 92.1, 9) = (Ed, 71.2, 5): False
//    (Abbey, 92.1, 9) = (Penelope, 82.9, 8): False
//    (Abbey, 92.1, 9) = (Ed, 71.2, 9): False
//    (Abbey, 92.1, 9) = (Judith, 84.3, 9): False
//
//    (Ed, 71.2, 9) = (Sam, 91.7, 8): False
//    (Ed, 71.2, 9) = (Ed, 71.2, 5): False
//    (Ed, 71.2, 9) = (Penelope, 82.9, 8): False
//    (Ed, 71.2, 9) = (Ed, 71.2, 9): True
//    (Ed, 71.2, 9) = (Judith, 84.3, 9): False
//
//    (Sam, 91.7, 8) = (Ed, 71.2, 5): False
//    (Sam, 91.7, 8) = (Penelope, 82.9, 8): False
//    (Sam, 91.7, 8) = (Ed, 71.2, 9): False
//    (Sam, 91.7, 8) = (Judith, 84.3, 9): False
//
//    (Ed, 71.2, 5) = (Penelope, 82.9, 8): False
//    (Ed, 71.2, 5) = (Ed, 71.2, 9): False
//    (Ed, 71.2, 5) = (Judith, 84.3, 9): False
//
//    (Penelope, 82.9, 8) = (Ed, 71.2, 9): False
//    (Penelope, 82.9, 8) = (Judith, 84.3, 9): False
```

```
//
//    (Ed, 71.2, 9) = (Judith, 84.3, 9): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2,T3> object.
- Its three components are of the same types as the current instance.
- Its three components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2,T3> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Tuple<T1,T2,T3>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3> object.

Examples

The following example illustrates the ToString method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
                      Tuple.Create("Penelope", 82.9, 8),
                      Tuple.Create("Linda", 99.0, 9),
                      Tuple.Create("Judith", 84.3, 9) };
      Array.Sort(scores);
      foreach (var score in scores)
         Console.WriteLine(score.ToString());
   }
```

```
}
// The example displays the following output;
//    (Abbey, 92.1, 9)
//    (Dave, 88.3, 9)
//    (Ed, 71.2, 5)
//    (Jack, 78.8, 8)
//    (Judith, 84.3, 9)
//    (Linda, 99, 9)
//    (Penelope, 82.9, 8)
//    (Sam, 91.7, 8)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*), where *Item1*, *Item2*, and *Item3* represent the values of the Item1, Item2, and Item3 properties, respectively. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

An object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3> object.

Examples

The following example creates an array of Tuple<T1,T2,T3> objects that consist of a student's name, mean test score, and number of tests. It displays the component of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display the value of each tuple in sorted order. To sort the array, the example defines a generic `ScoreComparer` class that implements the IComparer interface and sorts the Tuple<T1,T2,T3> objects in ascending order by the value of their second component rather than their first component. Note that the example does not directly call the Tuple<T1,T2,T3>.IStructuralComparable.CompareTo method. This method is called implicitly by the Array.Sort(Array, IComparer) method for each element in the array.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class ScoreComparer<T1, T2, T3> : IComparer
{
   public int Compare(object x, object y)
   {
      Tuple<T1, T2, T3> tX = x as Tuple<T1,T2, T3>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
         Tuple<T1, T2, T3> tY = y as Tuple<T1, T2, T3>;
         return Comparer<T2>.Default.Compare(tX.Item2, tY.Item2);
      }
   }
```

```
}

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                { Tuple.Create("Jack", 78.8, 8),
                  Tuple.Create("Abbey", 92.1, 9),
                  Tuple.Create("Dave", 88.3, 9),
                  Tuple.Create("Sam", 91.7, 8),
                  Tuple.Create("Ed", 71.2, 5),
                  Tuple.Create("Penelope", 82.9, 8),
                  Tuple.Create("Linda", 99.0, 9),
                  Tuple.Create("Judith", 84.3, 9) };

      Console.WriteLine("The values in unsorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());

      Console.WriteLine();

      Array.Sort(scores, new ScoreComparer<string, double, int>());

      Console.WriteLine("The values in sorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());
   }
}
// The example displays the following output;
//       The values in unsorted order:
//       (Jack, 78.8, 8)
//       (Abbey, 92.1, 9)
//       (Dave, 88.3, 9)
//       (Sam, 91.7, 8)
//       (Ed, 71.2, 5)
//       (Penelope, 82.9, 8)
//       (Linda, 99, 9)
//       (Judith, 84.3, 9)
//
//       The values in sorted order:
//       (Ed, 71.2, 5)
//       (Jack, 78.8, 8)
//       (Penelope, 82.9, 8)
//       (Judith, 84.3, 9)
//       (Dave, 88.3, 9)
//       (Sam, 91.7, 8)
//       (Abbey, 92.1, 9)
//       (Linda, 99, 9)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3> instance is cast to an IStructuralComparable interface.

Although this method can be called directly, it is most commonly called by collection sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **Tuple<T1,T2,T3>.IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Tuple<T1,T2,T3>.IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IStructuralEquatable.Equals(Object, IEqualityComparer)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Tuple<T1,T2,T3>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines an `Item2Comparer` class that implements the IEqualityComparer interface and changes the way in which Tuple<T1,T2,T3> objects are

evaluated for equality. The method always returns `true` when it is passed the Item1 property values of two Tuple<T1,T2,T3> objects, and it calls the Tuple<T1,T2,T3>.IStructuralEquatable.Equals method to evaluate their Item2 property values. If this method call returns `true`, their Item3 property values are passed to the method, which always returns `true`. As a result, the method tests for equality based only on the value of the Item2 property. The output illustrates the result for a data set of Tuple<T1,T2,T3> objects that record the names, mean test score, and number of tests of students in a class.

C#

```
using System;
using System.Collections;

public class Item2Comparer<T1, T2, T3> : IEqualityComparer
{
   new public bool Equals(object x, object y)
   {
      // Return true for all values of Item1.
      if (x is T1)
         return true;
      else if (x is T2)
         return x.Equals(y);
      else
         return true;
   }

   public int GetHashCode(object obj)
   {
      if (obj is T1)
         return ((T1) obj).GetHashCode();
      else if (obj is T2)
         return ((T2) obj).GetHashCode();
      else
         return ((T3) obj).GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
           { Tuple.Create("Ed", 78.8, 8),
             Tuple.Create("Abbey", 92.1, 9),
             Tuple.Create("Jim", 71.2, 9),
             Tuple.Create("Sam", 91.7, 8),
             Tuple.Create("Sandy", 71.2, 5),
             Tuple.Create("Penelope", 82.9, 8),
             Tuple.Create("Serena", 71.2, 9),
             Tuple.Create("Judith", 84.3, 9) };
```

```
      for (int ctr = 0; ctr < scores.Length; ctr++)
      {
         IStructuralEquatable score = scores[ctr];
         for (int ctr2 = ctr + 1; ctr2 < scores.Length; ctr2++)
         {
            Console.WriteLine("{0} = {1}: {2}", score,
                              scores[ctr2],
                              score.Equals(scores[ctr2],
                                           new Item2Comparer<string, double,
int>()));
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//      (Ed, 78.8, 8) = (Abbey, 92.1, 9): False
//      (Ed, 78.8, 8) = (Jim, 71.2, 9): False
//      (Ed, 78.8, 8) = (Sam, 91.7, 8): False
//      (Ed, 78.8, 8) = (Sandy, 71.2, 5): False
//      (Ed, 78.8, 8) = (Penelope, 82.9, 8): False
//      (Ed, 78.8, 8) = (Serena, 71.2, 9): False
//      (Ed, 78.8, 8) = (Judith, 84.3, 9): False
//
//      (Abbey, 92.1, 9) = (Jim, 71.2, 9): False
//      (Abbey, 92.1, 9) = (Sam, 91.7, 8): False
//      (Abbey, 92.1, 9) = (Sandy, 71.2, 5): False
//      (Abbey, 92.1, 9) = (Penelope, 82.9, 8): False
//      (Abbey, 92.1, 9) = (Serena, 71.2, 9): False
//      (Abbey, 92.1, 9) = (Judith, 84.3, 9): False
//
//      (Jim, 71.2, 9) = (Sam, 91.7, 8): False
//      (Jim, 71.2, 9) = (Sandy, 71.2, 5): True
//      (Jim, 71.2, 9) = (Penelope, 82.9, 8): False
//      (Jim, 71.2, 9) = (Serena, 71.2, 9): True
//      (Jim, 71.2, 9) = (Judith, 84.3, 9): False
//
//      (Sam, 91.7, 8) = (Sandy, 71.2, 5): False
//      (Sam, 91.7, 8) = (Penelope, 82.9, 8): False
//      (Sam, 91.7, 8) = (Serena, 71.2, 9): False
//      (Sam, 91.7, 8) = (Judith, 84.3, 9): False
//
//      (Sandy, 71.2, 5) = (Penelope, 82.9, 8): False
//      (Sandy, 71.2, 5) = (Serena, 71.2, 9): True
//      (Sandy, 71.2, 5) = (Judith, 84.3, 9): False
//
//      (Penelope, 82.9, 8) = (Serena, 71.2, 9): False
//      (Penelope, 82.9, 8) = (Judith, 84.3, 9): False
//
//      (Serena, 71.2, 9) = (Judith, 84.3, 9): False
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3> object whose components are of the same types as the current instance. The Tuple<T1,T2,T3>.IStructuralEquatable.Equals method first passes the Item1 values of the Tuple<T1,T2,T3> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2,T3> objects. If this method call returns `true` again, the method is called a third time and passed the Item3 values of the two Tuple<T1,T2,T3> objects.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

comparer IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3> object.

Examples

The following example creates an array of Tuple<T1,T2,T3> objects whose components consist of a student's name, mean test score, and number of tests. It displays the components of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display each tuple in sorted order. The output shows that the array has been sorted by its first component. Note that the example does not directly call the Tuple<T1,T2,T3>.IComparable.CompareTo method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, double, int>[] scores =
                    { Tuple.Create("Jack", 78.8, 8),
                      Tuple.Create("Abbey", 92.1, 9),
                      Tuple.Create("Dave", 88.3, 9),
                      Tuple.Create("Sam", 91.7, 8),
                      Tuple.Create("Ed", 71.2, 5),
                      Tuple.Create("Penelope", 82.9, 8),
                      Tuple.Create("Linda", 99.0, 9),
                      Tuple.Create("Judith", 84.3, 9) };

      Console.WriteLine("The values in unsorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());

      Console.WriteLine();

      Array.Sort(scores);

      Console.WriteLine("The values in sorted order:");
      foreach (var score in scores)
         Console.WriteLine(score.ToString());
   }
}
```

```
// The example displays the following output;
//    The values in unsorted order:
//    (Jack, 78.8, 8)
//    (Abbey, 92.1, 9)
//    (Dave, 88.3, 9)
//    (Sam, 91.7, 8)
//    (Ed, 71.2, 5)
//    (Penelope, 82.9, 8)
//    (Linda, 99, 9)
//    (Judith, 84.3, 9)
//
//    The values in sorted order:
//    (Abbey, 92.1, 9)
//    (Dave, 88.3, 9)
//    (Ed, 71.2, 5)
//    (Jack, 78.8, 8)
//    (Judith, 84.3, 9)
//    (Linda, 99, 9)
//    (Penelope, 82.9, 8)
//    (Sam, 91.7, 8)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3> class. Although the method can be called directly, it is most commonly called by the default overloads of collection sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **Tuple<T1,T2,T3>.IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Equals** method.

The Tuple<T1,T2,T3>.IComparable.CompareTo method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 2.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 2.

Remarks

The Tuple<T1,T2,T3>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

C#

```
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

3, the number of elements in a Tuple<T1,T2,T3> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 4-tuple, or quadruple.

C#

```
public class Tuple<T1,T2,T3,T4> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

T4

The type of the tuple's fourth component.

Inheritance Object → Tuple<T1,T2,T3,T4>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3,T4> class represents a 4-tuple, or quadruple, which is a tuple that has

four components.

You can instantiate a Tuple<T1,T2,T3,T4> object by calling either the Tuple<T1,T2,T3,T4> constructor or the static Tuple.Create<T1,T2,T3,T4>(T1, T2, T3, T4) method. You can retrieve the value of the tuple's components by using the read-only Item1, Item2, Item3, and Item4 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set. The following example defines an array of Tuple<T1,T2,T3,T4> objects that contain the names of baseball pitchers, the number of innings they pitched, and the number of earned runs (runs that scored without fielding errors), and hits that they gave up. The array is passed to the `ComputeStatistics` method, which calculates each pitcher's earned run average (the average number of runs given up in a nine-inning game), and the average number of hits given up per inning. The method also uses these two averages to compute a hypothetical effectiveness average.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Tuple<string, decimal, int, int>[] pitchers  =
           { Tuple.Create("McHale, Joe", 240.1m, 221, 96),
             Tuple.Create("Paul, Dave", 233.1m, 231, 84),
             Tuple.Create("Williams, Mike", 193.2m, 183, 86),
             Tuple.Create("Blair, Jack", 168.1m, 146, 65),
             Tuple.Create("Henry, Walt", 140.1m, 96, 30),
             Tuple.Create("Lee, Adam", 137.2m, 109, 45),
             Tuple.Create("Rohr, Don", 101.0m, 110, 42) };
      Tuple<string, double, double, double>[] results=
ComputeStatistics(pitchers);

      // Display the results.
      Console.WriteLine("{0,-20} {1,9} {2,11} {3,15}\n",
                        "Pitcher", "ERA", "Hits/Inn.",
"Effectiveness");
      foreach (var result in results)
         Console.WriteLine("{0,-20} {1,9:F2} {2,11:F2} {3,15:F2}",
                        result.Item1, result.Item2, result.Item3,
result.Item4);
```

```
    }

    private static Tuple<string, double, double, double>[]
ComputeStatistics(Tuple<string, decimal, int, int>[] pitchers)
    {
        var list = new List<Tuple<string, double, double, double>>();
        Tuple<string, double, double, double> result;

        foreach (var pitcher in pitchers)
        {
            // Decimal portion of innings pitched represents 1/3 of an
inning
            double innings = (double) Math.Truncate(pitcher.Item2);
            innings = innings + (((double)pitcher.Item2 - innings) * .33);

            double ERA = pitcher.Item4/innings * 9;
            double hitsPerInning = pitcher.Item3/innings;
            double EI = (ERA * 2 + hitsPerInning * 9)/3;
            result = new Tuple<string, double, double, double>
                                (pitcher.Item1, ERA, hitsPerInning, EI);
            list.Add(result);
        }
        return list.ToArray();
    }
}
// The example displays the following output;
//       Pitcher                     ERA   Hits/Inn.   Effectiveness
//
//       McHale, Joe               3.60        0.92            5.16
//       Paul, Dave                3.24        0.99            5.14
//       Williams, Mike            4.01        0.95            5.52
//       Blair, Jack               3.48        0.87            4.93
//       Henry, Walt               1.93        0.69            3.34
//       Lee, Adam                 2.95        0.80            4.36
//       Rohr, Don                 3.74        1.09            5.76
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its computed statistics, along with the name of the pitcher, in an array of Tuple<T1,T2,T3,T4> objects.

- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2,T3,T4> object as the method argument, you can supply the thread's startup routine with four items of data.

Constructors

Expand table

Tuple<T1,T2,T3,T4>(T1, T2, T3, T4)	Initializes a new instance of the Tuple<T1,T2,T3,T4> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3,T4> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3,T4> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3,T4> object's third component.
Item4	Gets the value of the current Tuple<T1,T2,T3,T4> object's fourth component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2,T3,T4> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3,T4> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current Tuple<T1,T2,T3,T4> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

IStructuralComparable.CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3,T4> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current Tuple<T1,T2,T3,T4> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>, T1, T2, T3, T4)	Deconstructs a tuple with 4 elements into separate variables.
ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>

- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>(T1, T2, T3, T4) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3,T4> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3, T4 item4);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

item4 T4

The value of the tuple's fourth component.

Remarks

You can also use the static Tuple.Create<T1,T2,T3,T4>(T1, T2, T3, T4) method to instantiate a 4-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2,T3,T4>(T1, T2, T3, T4) method to instantiate a 4-tuple whose components are of type String, Double, Double, and Double.

C#

```
var tuple4 = Tuple.Create("New York", 32.68, 51.87, 76.3);
Console.WriteLine("{0}: Hi {1}, Lo {2}, Ave {3}",
                  tuple4.Item1, tuple4.Item4, tuple4.Item2,
                  tuple4.Item3);
// Displays New York: Hi 76.3, Lo 32.68, Ave 51.87
```

This is equivalent to the following call to the Tuple<T1,T2,T3,T4> class constructor.

C#

```
var tuple4 = new Tuple<string, double, double, double>
                      ("New York", 32.68, 51.87, 76.3);
Console.WriteLine("{0}: Hi {1}, Lo {2}, Ave {3}",
                  tuple4.Item1, tuple4.Item4, tuple4.Item2,
                  tuple4.Item3);
// Displays New York: Hi 76.3, Lo 32.68, Ave 51.87
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3,T4> object's first component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4> objects whose components contain the name of a city, a month of the year, and the average high and low temperatures for that month. It then retrieves and displays the value of each component.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      Tuple<string, int, double, double>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61.0, 43.0),
              Tuple.Create("Chicago, IL", 2, 34.0, 18.0),
              Tuple.Create("Newark, NJ", 4, 61.0, 43.0),
              Tuple.Create("Boston, MA", 6, 77.0, 59.0),
              Tuple.Create("Detroit, MI", 9, 74.0, 53.0),
              Tuple.Create("Minneapolis, MN", 8, 81.0, 61.0) };
      // Display the array of 4-tuple objects.
```

```
      Console.WriteLine("{0,41}", "Temperatures");
      Console.WriteLine("{0,-20} {1,5}    {2,4}  {3,4}\n",
                        "City", "Month", "High", "Low");
      foreach (var temperature in temperatures)
         Console.WriteLine("{0,-20} {1,5}    {2,4:N1}  {3,4:N1}",
                           temperature.Item1,

DateTimeFormatInfo.CurrentInfo.GetAbbreviatedMonthName(temperature.Item2 -
1),
                           temperature.Item3, temperature.Item4);
   }
}
// The example displays the following output:
//                                     Temperatures
//       City                 Month    High   Low
//
//       New York, NY           Mar    61.0  43.0
//       Chicago, IL            Jan    34.0  18.0
//       Newark, NJ             Mar    61.0  43.0
//       Boston, MA             May    77.0  59.0
//       Detroit, MI            Aug    74.0  53.0
//       Minneapolis, MN        Jul    81.0  61.0
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3,T4> object's second component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4> objects whose components contain the name of a city, a month of the year, and the average high and low temperatures for that month. It then retrieves and displays the value of each component.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      Tuple<string, int, double, double>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61.0, 43.0),
              Tuple.Create("Chicago, IL", 2, 34.0, 18.0),
              Tuple.Create("Newark, NJ", 4, 61.0, 43.0),
              Tuple.Create("Boston, MA", 6, 77.0, 59.0),
              Tuple.Create("Detroit, MI", 9, 74.0, 53.0),
              Tuple.Create("Minneapolis, MN", 8, 81.0, 61.0) };
      // Display the array of 4-tuple objects.
```

```
      Console.WriteLine("{0,41}", "Temperatures");
      Console.WriteLine("{0,-20} {1,5}    {2,4}  {3,4}\n",
                        "City", "Month", "High", "Low");
      foreach (var temperature in temperatures)
         Console.WriteLine("{0,-20} {1,5}    {2,4:N1}  {3,4:N1}",
                           temperature.Item1,

DateTimeFormatInfo.CurrentInfo.GetAbbreviatedMonthName(temperature.Item2 -
1),
                           temperature.Item3, temperature.Item4);
   }
}
// The example displays the following output:
//                                     Temperatures
//       City                 Month    High   Low
//
//       New York, NY           Mar    61.0  43.0
//       Chicago, IL            Jan    34.0  18.0
//       Newark, NJ             Mar    61.0  43.0
//       Boston, MA             May    77.0  59.0
//       Detroit, MI            Aug    74.0  53.0
//       Minneapolis, MN        Jul    81.0  61.0
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3,T4> object's third component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4> objects whose components contain the name of a city, a month of the year, and the average high and low temperatures for that month. It then retrieves and displays the value of each component.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      Tuple<string, int, double, double>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61.0, 43.0),
              Tuple.Create("Chicago, IL", 2, 34.0, 18.0),
              Tuple.Create("Newark, NJ", 4, 61.0, 43.0),
              Tuple.Create("Boston, MA", 6, 77.0, 59.0),
              Tuple.Create("Detroit, MI", 9, 74.0, 53.0),
              Tuple.Create("Minneapolis, MN", 8, 81.0, 61.0) };
      // Display the array of 4-tuple objects.
```

```
      Console.WriteLine("{0,41}", "Temperatures");
      Console.WriteLine("{0,-20} {1,5}    {2,4}  {3,4}\n",
                        "City", "Month", "High", "Low");
      foreach (var temperature in temperatures)
         Console.WriteLine("{0,-20} {1,5}    {2,4:N1}  {3,4:N1}",
                           temperature.Item1,

DateTimeFormatInfo.CurrentInfo.GetAbbreviatedMonthName(temperature.Item2 -
1),
                           temperature.Item3, temperature.Item4);
   }
}
// The example displays the following output:
//                                     Temperatures
//       City                  Month    High   Low
//
//       New York, NY            Mar    61.0  43.0
//       Chicago, IL             Jan    34.0  18.0
//       Newark, NJ              Mar    61.0  43.0
//       Boston, MA              May    77.0  59.0
//       Detroit, MI             Aug    74.0  53.0
//       Minneapolis, MN         Jul    81.0  61.0
```

Remarks

You can dynamically determine the type of the Item3 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item3 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the third element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4>.Item4 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4> object's fourth component.

C#

```
public T4 Item4 { get; }
```

Property Value

T4

The value of the current Tuple<T1,T2,T3,T4> object's fourth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4> objects whose components contain the name of a city, a month of the year, and the average high and low temperatures for that month. It then retrieves and displays the value of each component.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      Tuple<string, int, double, double>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61.0, 43.0),
              Tuple.Create("Chicago, IL", 2, 34.0, 18.0),
              Tuple.Create("Newark, NJ", 4, 61.0, 43.0),
              Tuple.Create("Boston, MA", 6, 77.0, 59.0),
              Tuple.Create("Detroit, MI", 9, 74.0, 53.0),
              Tuple.Create("Minneapolis, MN", 8, 81.0, 61.0) };
      // Display the array of 4-tuple objects.
```

```
      Console.WriteLine("{0,41}", "Temperatures");
      Console.WriteLine("{0,-20} {1,5}    {2,4}  {3,4}\n",
                        "City", "Month", "High", "Low");
      foreach (var temperature in temperatures)
         Console.WriteLine("{0,-20} {1,5}    {2,4:N1}  {3,4:N1}",
                           temperature.Item1,

DateTimeFormatInfo.CurrentInfo.GetAbbreviatedMonthName(temperature.Item2 -
1),
                           temperature.Item3, temperature.Item4);
   }
}
// The example displays the following output:
//                                     Temperatures
//       City                  Month    High   Low
//
//       New York, NY            Mar    61.0  43.0
//       Chicago, IL             Jan    34.0  18.0
//       Newark, NJ              Mar    61.0  43.0
//       Boston, MA              May    77.0  59.0
//       Detroit, MI             Aug    74.0  53.0
//       Minneapolis, MN         Jul    81.0  61.0
```

Remarks

You can dynamically determine the type of the Item4 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item4 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the fourth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4> object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4> objects that provide data on temperatures at three times during a particular day. The Equals(Object) method is called to compare every Tuple<T1,T2,T3,T4> object with every other Tuple<T1,T2,T3,T4> object. The output illustrates that the Equals(Object) method returns `true` only when all four components of the Tuple<T1,T2,T3,T4> objects have equal values.

C#

```
using System;

public class Class1
```

```
{
   public static void Main()
   {
      Tuple<DateTime, double, double, double>[] temperatures =
            { Tuple.Create(new DateTime(2009, 1, 16), 3.0, 5.0, 4.0),
              Tuple.Create(new DateTime(2009, 4, 22), 9.0, 14.0, 11.0),
              Tuple.Create(new DateTime(2009, 4, 22), 9.0, 14.0, 10.0),
              Tuple.Create(new DateTime(2009, 6, 1), 23.0, 28.0, 21.0),
              Tuple.Create(new DateTime(2009, 4, 22), 9.0, 14.0, 11.0),
              Tuple.Create(new DateTime(2009, 9, 6), 25.0, 30.0, 25.0) };
      // Compare each item with every other item for equality.
      for (int ctr = 0; ctr < temperatures.Length; ctr++)
      {
         var temperatureInfo = temperatures[ctr];
         for (int ctr2 = ctr + 1; ctr2 < temperatures.Length; ctr2++)
            Console.WriteLine("{0} = {1}: {2}", temperatureInfo,
temperatures[ctr2],

temperatureInfo.Equals(temperatures[ctr2]));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    (1/16/2009 12:00:00 AM, 3, 5, 4) = (4/22/2009 12:00:00 AM, 9, 14, 11):
False
//    (1/16/2009 12:00:00 AM, 3, 5, 4) = (4/22/2009 12:00:00 AM, 9, 14, 10):
False
//    (1/16/2009 12:00:00 AM, 3, 5, 4) = (6/1/2009 12:00:00 AM, 23, 28, 21):
False
//    (1/16/2009 12:00:00 AM, 3, 5, 4) = (4/22/2009 12:00:00 AM, 9, 14, 11):
False
//    (1/16/2009 12:00:00 AM, 3, 5, 4) = (9/6/2009 12:00:00 AM, 25, 30, 25):
False
//
//    (4/22/2009 12:00:00 AM, 9, 14, 11) = (4/22/2009 12:00:00 AM, 9, 14,
10): False
//    (4/22/2009 12:00:00 AM, 9, 14, 11) = (6/1/2009 12:00:00 AM, 23, 28,
21): False
//    (4/22/2009 12:00:00 AM, 9, 14, 11) = (4/22/2009 12:00:00 AM, 9, 14,
11): True
//    (4/22/2009 12:00:00 AM, 9, 14, 11) = (9/6/2009 12:00:00 AM, 25, 30,
25): False
//
//    (4/22/2009 12:00:00 AM, 9, 14, 10) = (6/1/2009 12:00:00 AM, 23, 28,
21): False
//    (4/22/2009 12:00:00 AM, 9, 14, 10) = (4/22/2009 12:00:00 AM, 9, 14,
11): False
//    (4/22/2009 12:00:00 AM, 9, 14, 10) = (9/6/2009 12:00:00 AM, 25, 30,
25): False
//
//    (6/1/2009 12:00:00 AM, 23, 28, 21) = (4/22/2009 12:00:00 AM, 9, 14,
11): False
//    (6/1/2009 12:00:00 AM, 23, 28, 21) = (9/6/2009 12:00:00 AM, 25, 30,
25): False
```

```
//
//    (4/22/2009 12:00:00 AM, 9, 14, 11) = (9/6/2009 12:00:00 AM, 25, 30,
25): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2,T3,T4> object.
- Its four components are of the same types as the current instance.
- Its four components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2,T3,T4> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3,T4> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3,T4> object.

Examples

The following example illustrates the ToString method. It displays the components of an array of 4-tuple objects that contain the name of a city, a month of the year, and the high and low average temperature for that month.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      Tuple<string, int, int, int>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61, 43),
              Tuple.Create("Chicago, IL", 2, 34, 18),
              Tuple.Create("Newark, NJ", 4, 61, 43),
              Tuple.Create("Boston, MA", 6, 77, 59),
              Tuple.Create("Detroit, MI", 9, 74, 53),
              Tuple.Create("Minneapolis, MN", 8, 81, 61) };
      // Display the array of 4-tuple objects.
      foreach (var temperature in temperatures)
         Console.WriteLine(temperature.ToString());
```

```
    }
}
// The example displays the following output:
//       (New York, NY, 4, 61, 43)
//       (Chicago, IL, 2, 34, 18)
//       (Newark, NJ, 4, 61, 43)
//       (Boston, MA, 6, 77, 59)
//       (Detroit, MI, 9, 74, 53)
//       (Minneapolis, MN, 8, 81, 61)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*), where *Item1*, *Item2*, *Item3*, and *Item4* represent the values of the Item1, Item2, Item3, and Item4 properties, respectively. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

An object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3,T4> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4> objects that contain statistical data about baseball pitchers. The data items include the name of the pitcher, the number of innings pitched, the pitcher's earned run average (the average number of runs a pitcher allows per game), and the number of hits the pitcher has given up. The example displays the component of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display the value of each tuple in sorted order. To sort the array, the example defines a generic `PitcherComparer` class that implements the IComparer interface and sorts the Tuple<T1,T2,T3,T4> objects in ascending order by the value of their third component (the earned run average) rather than their first component. Note that the example does not directly call the IStructuralComparable.CompareTo(Object, IComparer) method. This method is called implicitly by the Array.Sort(Array, IComparer) method for each element in the array.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class PitcherComparer<T1, T2, T3, T4> : IComparer
{
   public int Compare(object x, object y)
   {
      Tuple<T1, T2, T3, T4> tX = x as Tuple<T1, T2, T3, T4>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
```

```
            Tuple<T1, T2, T3, T4> tY = y as Tuple<T1, T2, T3, T4>;
            return Comparer<T3>.Default.Compare(tX.Item3, tY.Item3);
         }
      }
   }

   public class Example
   {
      public static void Main()
      {
         Tuple<string, double, double, int>[] pitchers =
                       { Tuple.Create("McHale, Joe", 240.1, 3.60, 221),
                         Tuple.Create("Paul, Dave", 233.1, 3.24, 231),
                         Tuple.Create("Williams, Mike", 193.2, 4.00, 183),
                         Tuple.Create("Blair, Jack", 168.1, 3.48, 146),
                         Tuple.Create("Henry, Walt", 140.1, 1.92, 96),
                         Tuple.Create("Lee, Adam", 137.2, 2.94, 109),
                         Tuple.Create("Rohr, Don", 101.0, 3.74, 110) };

         Console.WriteLine("The values in unsorted order:");
         foreach (var pitcher in pitchers)
            Console.WriteLine(pitcher.ToString());

         Console.WriteLine();

         Array.Sort(pitchers, new PitcherComparer<string, double, double, int>
());

         Console.WriteLine("The values sorted by earned run average (component
3):");
         foreach (var pitcher in pitchers)
            Console.WriteLine(pitcher.ToString());
      }
   }
   // The example displays the following output;
   //       The values in unsorted order:
   //       (McHale, Joe, 240.1, 3.6, 221)
   //       (Paul, Dave, 233.1, 3.24, 231)
   //       (Williams, Mike, 193.2, 4, 183)
   //       (Blair, Jack, 168.1, 3.48, 146)
   //       (Henry, Walt, 140.1, 1.92, 96)
   //       (Lee, Adam, 137.2, 2.94, 109)
   //       (Rohr, Don, 101, 3.74, 110)
   //
   //       The values sorted by earned run average (component 3):
   //       (Henry, Walt, 140.1, 1.92, 96)
   //       (Lee, Adam, 137.2, 2.94, 109)
   //       (Rohr, Don, 101, 3.74, 110)
   //       (Blair, Jack, 168.1, 3.48, 146)
   //       (McHale, Joe, 240.1, 3.6, 221)
   //       (Paul, Dave, 233.1, 3.24, 231)
   //       (Williams, Mike, 193.2, 4, 183)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4> instance is cast to an IStructuralComparable interface.

Although this method can be called directly, it is most commonly called by collection sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo(Object, IComparer)** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals(Object, IEqualityComparer)** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

`other` Object

The object to compare with this instance.

`comparer` IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines an `Item3And4Comparer` class that implements the IEqualityComparer interface and changes the way in which Tuple<T1,T2,T3,T4> objects

are evaluated for equality. The method always returns `true` when it is passed the Item1 and Item2 property values of two Tuple<T1,T2,T3,T4> objects, and it calls the `obj.Equals` method to evaluate their Item3 property values. It this method call returns true, it also calls the `obj.Equals` method to evaluate the tuples' Item4 property values. As a result, the method tests for equality based only on the values of the Item3 and Item4 properties. The output illustrates the result for a data set of Tuple<T1,T2,T3,T4> objects that record the name of a U.S. city, the month of a year, and the average high and low temperature for that month.

C#

```
using System;
using System.Collections;

public class Item3And4Comparer<T1, T2, T3, T4> : IEqualityComparer
{
   private int argument = 0;

   new public bool Equals(object x, object y)
   {
      argument++;

      // Return true for all values of Item1, Item2.
      if (argument <= 2)
         return true;
      else
         return x.Equals(y);
   }

   public int GetHashCode(object obj)
   {
      if (obj is T1)
         return ((T1) obj).GetHashCode();
      else if (obj is T2)
         return ((T2) obj).GetHashCode();
      else if (obj is T3)
         return ((T3) obj).GetHashCode();
      else
         return ((T4) obj).GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      Tuple<string, int, double, double>[] temperatures =
            { Tuple.Create("New York, NY", 4, 61.0, 43.0),
              Tuple.Create("Chicago, IL", 2, 34.0, 18.0),
              Tuple.Create("Newark, NJ", 4, 61.0, 43.0),
              Tuple.Create("Boston, MA", 6, 77.0, 59.0),
```

```
                Tuple.Create("Detroit, MI", 9, 74.0, 53.0),
                Tuple.Create("Minneapolis, MN", 8, 81.0, 61.0) };
        // Compare each item with every other item for equality.
        for (int ctr = 0; ctr < temperatures.Length; ctr++)
        {
           IStructuralEquatable temperatureInfo = temperatures[ctr];
           for (int ctr2 = ctr + 1; ctr2 < temperatures.Length; ctr2++)
              Console.WriteLine("{0} = {1}: {2}",
                                temperatureInfo, temperatures[ctr2],
                                temperatureInfo.Equals(temperatures[ctr2],
                                                       new Item3And4Comparer<string,
int, double, double>()));

           Console.WriteLine();
        }
    }
}
// The example displays the following output:
//    (New York, NY, 4, 61, 43) = (Chicago, IL, 2, 34, 18): False
//    (New York, NY, 4, 61, 43) = (Newark, NJ, 4, 61, 43): True
//    (New York, NY, 4, 61, 43) = (Boston, MA, 6, 77, 59): False
//    (New York, NY, 4, 61, 43) = (Detroit, MI, 9, 74, 53): False
//    (New York, NY, 4, 61, 43) = (Minneapolis, MN, 8, 81, 61): False
//
//    (Chicago, IL, 2, 34, 18) = (Newark, NJ, 4, 61, 43): False
//    (Chicago, IL, 2, 34, 18) = (Boston, MA, 6, 77, 59): False
//    (Chicago, IL, 2, 34, 18) = (Detroit, MI, 9, 74, 53): False
//    (Chicago, IL, 2, 34, 18) = (Minneapolis, MN, 8, 81, 61): False
//
//    (Newark, NJ, 4, 61, 43) = (Boston, MA, 6, 77, 59): False
//    (Newark, NJ, 4, 61, 43) = (Detroit, MI, 9, 74, 53): False
//    (Newark, NJ, 4, 61, 43) = (Minneapolis, MN, 8, 81, 61): False
//
//    (Boston, MA, 6, 77, 59) = (Detroit, MI, 9, 74, 53): False
//    (Boston, MA, 6, 77, 59) = (Minneapolis, MN, 8, 81, 61): False
//
//    (Detroit, MI, 9, 74, 53) = (Minneapolis, MN, 8, 81, 61): False
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3,T4> object whose components are of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the Tuple<T1,T2,T3,T4> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method

is called again and passed the Item2 values of the two Tuple<T1,T2,T3,T4> objects. If this method call returns `true` again, the method is called a third time and passed the Item3 values of the two Tuple<T1,T2,T3,T4> objects. If this method call returns `true` again, the method is called for the fourth and final time and passed the Item4 values of the two Tuple<T1,T2,T3,T4> objects.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.IStructural Equatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3,T4> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3,T4> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3,T4> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4> objects whose components consist of a baseball pitcher's name, number of innings pitched, and number of hits and earned runs given up. It displays the components of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display each tuple in sorted order. The output shows that the array has been sorted by name, which is the first component. Note that the example does not directly call the IComparable.CompareTo(Object) method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Tuple<string, decimal, int, int>[] pitchers  =
                        { Tuple.Create("McHale, Joe", 240.1m, 221, 96),
                          Tuple.Create("Paul, Dave", 233.1m, 231, 84),
                          Tuple.Create("Williams, Mike", 193.2m, 183, 86),
                          Tuple.Create("Blair, Jack", 168.1m, 146, 65),
                          Tuple.Create("Henry, Walt", 140.1m, 96, 30),
                          Tuple.Create("Lee, Adam", 137.2m, 109, 45),
                          Tuple.Create("Rohr, Don", 101.0m, 110, 42) };

      // Display the array in unsorted order.
      Console.WriteLine("The values in unsorted order:");
      foreach (var pitcher in pitchers)
         Console.WriteLine(pitcher.ToString());
      Console.WriteLine();

      // Sort the array
      Array.Sort(pitchers);

      // Display the array in sorted order.
      Console.WriteLine("The values in sorted order:");
      foreach (var pitcher in pitchers)
```

```
        Console.WriteLine(pitcher.ToString());
    }
}
// The example displays the following output;
//       The values in unsorted order:
//       (McHale, Joe, 240.1, 221, 96)
//       (Paul, Dave, 233.1, 231, 84)
//       (Williams, Mike, 193.2, 183, 86)
//       (Blair, Jack, 168.1, 146, 65)
//       (Henry, Walt, 140.1, 96, 30)
//       (Lee, Adam, 137.2, 109, 45)
//       (Rohr, Don, 101, 110, 42)
//
//       The values in sorted order:
//       (Blair, Jack, 168.1, 146, 65)
//       (Henry, Walt, 140.1, 96, 30)
//       (Lee, Adam, 137.2, 109, 45)
//       (McHale, Joe, 240.1, 221, 96)
//       (Paul, Dave, 233.1, 231, 84)
//       (Rohr, Don, 101, 110, 42)
//       (Williams, Mike, 193.2, 183, 86)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3,T4> class. Although the method can be called directly, it is most commonly called by the default overloads of collection sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Equals** method.

The IComparable.CompareTo method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 3.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 3.

Remarks

The Tuple<T1,T2,T3,T4>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4> object to an ITuple interface

object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

4, the number of elements in a Tuple<T1,T2,T3,T4> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 5-tuple, or quintuple.

C#

```
public class Tuple<T1,T2,T3,T4,T5> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

T4

The type of the tuple's fourth component.

T5

The type of the tuple's fifth component.

Inheritance Object → Tuple<T1,T2,T3,T4,T5>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3,T4,T5> class represents a 5-tuple, or quintuple, which is a tuple that has five components.

You can instantiate a Tuple<T1,T2,T3,T4,T5> object by calling either the Tuple<T1,T2,T3,T4,T5> constructor or the static Tuple.Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5) method. You can retrieve the value of the tuple's components by using the read-only Item1, Item2, Item3, Item4, and Item5 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set. The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects that contain the names of running backs in American football, the number of games in which they played, and the number of carries, total yards gained, and touchdowns scored during those games. The array is passed to the `ComputeStatistics` method, which calculates each running back's number of carries per game, average yards per game, average yards per carry, and average number of touchdowns per attempt.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      // Organization of runningBacks 5-tuple:
      //    Component 1: Player name
      //    Component 2: Number of games played
      //    Component 3: Number of attempts (carries)
      //    Component 4: Number of yards gained
      //    Component 5: Number of touchdowns
      Tuple<string, int, int, int, int>[] runningBacks =
           { Tuple.Create("Payton, Walter", 190, 3838, 16726, 110),
             Tuple.Create("Sanders, Barry", 153, 3062, 15269, 99),
             Tuple.Create("Brown, Jim", 118, 2359, 12312, 106),
             Tuple.Create("Dickerson, Eric", 144, 2996, 13259, 90),
             Tuple.Create("Faulk, Marshall", 176, 2836, 12279, 100) };
      // Calculate statistics.
      // Organization of runningStats 5-tuple:
      //    Component 1: Player name
      //    Component 2: Number of attempts per game
      //    Component 3: Number of yards per game
      //    Component 4: Number of yards per attempt
```

```
      //    Component 5: Number of touchdowns per attempt
      Tuple<string, double, double, double, double>[] runningStats  =
          ComputeStatistics(runningBacks);

      // Display the result.
      Console.WriteLine("{0,-16} {1,5} {2,6} {3,7} {4,7} {5,7} {6,7}
{7,5} {8,7}\n",
                        "Name", "Games", "Att", "Att/Gm", "Yards",
"Yds/Gm",
                        "Yds/Att", "TD", "TD/Att");
      for (int ctr = 0; ctr < runningBacks.Length; ctr++)
         Console.WriteLine("{0,-16} {1,5} {2,6:N0} {3,7:N1} {4,7:N0}
{5,7:N1} {6,7:N2} {7,5} {8,7:N3}\n",
                           runningBacks[ctr].Item1,
runningBacks[ctr].Item2, runningBacks[ctr].Item3,
                           runningStats[ctr].Item2,
runningBacks[ctr].Item4, runningStats[ctr].Item3,
                           runningStats[ctr].Item4,
runningBacks[ctr].Item5, runningStats[ctr].Item5);
   }

   private static Tuple<string, double, double, double, double>[]
ComputeStatistics(
                Tuple<string, int, int, int, int>[] players)
   {
      Tuple<string, double, double, double, double> result;
      var list = new List<Tuple<string, double, double, double,
double>>();

      foreach (var player in players)
      {
         // Create result object containing player name and statistics.
         result = Tuple.Create(player.Item1,
                               player.Item3/((double)player.Item2),
                               player.Item4/((double)player.Item2),
                               player.Item4/((double)player.Item3),
                               player.Item5/((double)player.Item3));
         list.Add(result);
      }
      return list.ToArray();
   }
}
// The example displays the following output:
//    Name             Games    Att  Att/Gm   Yards  Yds/Gm Yds/Att
TD  TD/Att
//
//    Payton, Walter     190  3,838    20.2  16,726    88.0    4.36
110   0.029
//
//    Sanders, Barry     153  3,062    20.0  15,269    99.8    4.99
99   0.032
//
//    Brown, Jim         118  2,359    20.0  12,312   104.3    5.22
106   0.045
//
```

```
//    Dickerson, Eric    144  2,996    20.8  13,259    92.1    4.43
90   0.030
//
//    Faulk, Marshall    176  2,836    16.1  12,279    69.8    4.33
100   0.035
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its computed statistics, along with the name of the player, in an array of Tuple<T1,T2,T3,T4,T5> objects.
- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2,T3,T4,T5> object as the method argument, you can supply the thread's startup routine with five items of data.

Constructors

Expand table

Tuple<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Initializes a new instance of the Tuple<T1,T2,T3,T4,T5> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's third component.
Item4	Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's fourth component.
Item5	Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's fifth component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2,T3,T4,T5> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5> instance.

Explicit Interface Implementations

Expand table

IComparable.Compare To(Object)	Compares the current Tuple<T1,T2,T3,T4,T5> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable. CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3,T4,T5> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable. Equals(Object, IEquality Comparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.Get HashCode(IEquality Comparer)	Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2,T3,T4,T5> (Tuple<T1,T2,T3,T4,T5>, T1, T2, T3, T4, T5)	Deconstructs a tuple with 5 elements into separate variables.

ToValueTuple<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3,T4,T5> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

item4 T4

The value of the tuple's fourth component.

item5 T5

The value of the tuple's fifth component.

Remarks

You can also use the static Tuple.Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5) method to instantiate a 5-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2,T3,T4,T5>(T1, T2, T3,

T4, T5) method to instantiate a 5-tuple whose first component is of type String and its remaining four components are of type Int32.

C#

```
var tuple5 = Tuple.Create("New York", 1990, 7322564, 2000, 8008278);
Console.WriteLine("{0}: {1:N0} in {2}, {3:N0} in {4}",
                  tuple5.Item1, tuple5.Item3, tuple5.Item2,
                  tuple5.Item5, tuple5.Item4);
// Displays New York: 7,322,564 in 1990, 8,008,278 in 2000
```

This is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5> class constructor.

C#

```
var tuple5 = new Tuple<string, int, int, int, int>
                      ("New York", 1990, 7322564, 2000, 8008278);
Console.WriteLine("{0}: {1:N0} in {2}, {3:N0} in {4}",
                  tuple5.Item1, tuple5.Item3, tuple5.Item2,
                  tuple5.Item5, tuple5.Item4);
// Displays New York: 7,322,564 in 1990, 8,008,278 in 2000
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3,T4,T5> object's first component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects whose components contain the name of a state in the United States, its population in 1990 and 2000, its population change in this 10-year period, and the percentage change in its population. It then iterates through the array and displays the value of each component in a tuple.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] statesData =
         { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
           Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
           Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
```

```
      // Display the items of each tuple
      Console.WriteLine("{0,-12}{1,18}{2,18}{3,15}{4,12}\n", "State",
                        "Population 1990", "Population 2000", "Change",
                        "% Change");
      foreach(Tuple<string, int, int, int, double> stateData in statesData)
         Console.WriteLine("{0,-12}{1,18:N0}{2,18:N0}{3,15:N0}{4,12:P1}",
                           stateData.Item1, stateData.Item2,
                           stateData.Item3, stateData.Item4,
stateData.Item5/100);
   }
}
// The example displays the following output:
//    State          Population 1990   Population 2000         Change    %
Change
//
//    California          29,760,021        33,871,648      4,111,627
13.8 %
//    Illinois            11,430,602        12,419,293        988,691
8.6 %
//    Washington           4,866,692         5,894,121      1,027,429
21.1 %
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3,T4,T5> object's second component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects whose components contain the name of a state in the United States, its population in 1990 and 2000, its population change in this 10-year period, and the percentage change in its population. It then iterates through the array and displays the value of each component in a tuple.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] statesData =
         { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
           Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
           Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
```

```
      // Display the items of each tuple
      Console.WriteLine("{0,-12}{1,18}{2,18}{3,15}{4,12}\n", "State",
                        "Population 1990", "Population 2000", "Change",
                        "% Change");
      foreach(Tuple<string, int, int, int, double> stateData in statesData)
         Console.WriteLine("{0,-12}{1,18:N0}{2,18:N0}{3,15:N0}{4,12:P1}",
                           stateData.Item1, stateData.Item2,
                           stateData.Item3, stateData.Item4,
stateData.Item5/100);
   }
}
// The example displays the following output:
//    State          Population 1990   Population 2000         Change    %
Change
//
//    California          29,760,021        33,871,648      4,111,627
13.8 %
//    Illinois            11,430,602        12,419,293        988,691
8.6 %
//    Washington           4,866,692         5,894,121      1,027,429
21.1 %
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3,T4,T5> object's third component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects whose components contain the name of a state in the United States, its population in 1990 and 2000, its population change in this 10-year period, and the percentage change in its population. It then iterates through the array and displays the value of each component in a tuple.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] statesData =
         { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
           Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
           Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
```

```
      // Display the items of each tuple
      Console.WriteLine("{0,-12}{1,18}{2,18}{3,15}{4,12}\n", "State",
                        "Population 1990", "Population 2000", "Change",
                        "% Change");
      foreach(Tuple<string, int, int, int, double> stateData in statesData)
         Console.WriteLine("{0,-12}{1,18:N0}{2,18:N0}{3,15:N0}{4,12:P1}",
                           stateData.Item1, stateData.Item2,
                           stateData.Item3, stateData.Item4,
stateData.Item5/100);
   }
}
// The example displays the following output:
//    State          Population 1990   Population 2000         Change    %
Change
//
//    California          29,760,021        33,871,648      4,111,627
13.8 %
//    Illinois            11,430,602        12,419,293        988,691
8.6 %
//    Washington           4,866,692         5,894,121      1,027,429
21.1 %
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Item4 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's fourth component.

C#

```
public T4 Item4 { get; }
```

Property Value

T4

The value of the current Tuple<T1,T2,T3,T4,T5> object's fourth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects whose components contain the name of a state in the United States, its population in 1990 and 2000, its population change in this 10-year period, and the percentage change in its population. It then iterates through the array and displays the value of each component in a tuple.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] statesData =
         { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
           Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
           Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
```

```
      // Display the items of each tuple
      Console.WriteLine("{0,-12}{1,18}{2,18}{3,15}{4,12}\n", "State",
                        "Population 1990", "Population 2000", "Change",
                        "% Change");
      foreach(Tuple<string, int, int, int, double> stateData in statesData)
         Console.WriteLine("{0,-12}{1,18:N0}{2,18:N0}{3,15:N0}{4,12:P1}",
                           stateData.Item1, stateData.Item2,
                           stateData.Item3, stateData.Item4,
stateData.Item5/100);
   }
}
// The example displays the following output:
//    State          Population 1990   Population 2000         Change    %
Change
//
//    California          29,760,021        33,871,648      4,111,627
13.8 %
//    Illinois            11,430,602        12,419,293        988,691
8.6 %
//    Washington           4,866,692         5,894,121      1,027,429
21.1 %
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Item5 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5> object's fifth component.

C#

```
public T5 Item5 { get; }
```

Property Value

T5

The value of the current Tuple<T1,T2,T3,T4,T5> object's fifth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5> objects whose components contain the name of a state in the United Dates, its population in 1990 and 2000, its population change in this 10-year period, and the percentage change in its population. It then iterates through the array and displays the value of each component in a tuple.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] statesData =
         { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
           Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
           Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
```

```
      // Display the items of each tuple
      Console.WriteLine("{0,-12}{1,18}{2,18}{3,15}{4,12}\n", "State",
                        "Population 1990", "Population 2000", "Change",
                        "% Change");
      foreach(Tuple<string, int, int, int, double> stateData in statesData)
         Console.WriteLine("{0,-12}{1,18:N0}{2,18:N0}{3,15:N0}{4,12:P1}",
                           stateData.Item1, stateData.Item2,
                           stateData.Item3, stateData.Item4,
stateData.Item5/100);
   }
}
// The example displays the following output:
//    State          Population 1990   Population 2000         Change    %
Change
//
//    California          29,760,021        33,871,648      4,111,627
13.8 %
//    Illinois            11,430,602        12,419,293        988,691
8.6 %
//    Washington           4,866,692         5,894,121      1,027,429
21.1 %
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5> object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example defines an array of 5-tuple objects that contain data about the temperatures of patients in two test groups. The first component of the array provides the number of the test group, and the second through fifth components provide the temperatures of a patient at hourly intervals. The Tuple<T1,T2,T3,T4,T5>.Equals(Object) method is called to compare every Tuple<T1,T2,T3,T4,T5> object with every other Tuple<T1,T2,T3,T4,T5> object. The output illustrates that the Equals method returns `true` only when all five components of the Tuple<T1,T2,T3,T4,T5> objects have equal values.

C#

```
using System;

public class Class1
{
   public static void Main()
   {
      Tuple<int, double, double, double, double>[] temperatureInfos =
                              { Tuple.Create(2, 97.9, 97.8, 98.0, 98.2),
                                Tuple.Create(1, 98.6, 98.8, 98.8, 99.0),
                                Tuple.Create(2, 98.6, 98.6, 98.6, 98.4),
                                Tuple.Create(1, 98.4, 98.6, 99.0, 99.2),
                                Tuple.Create(2, 98.6, 98.6, 98.6, 98.4),
                                Tuple.Create(1, 98.6, 98.8, 98.8, 99.0) };
      // Compare each item with every other item for equality.
      for (int ctr = 0; ctr < temperatureInfos.Length; ctr++)
      {
         var temperatureInfo = temperatureInfos[ctr];
         for (int ctr2 = ctr + 1; ctr2 < temperatureInfos.Length; ctr2++)
            Console.WriteLine("{0} = {1}: {2}", temperatureInfo,
temperatureInfos[ctr2],

temperatureInfo.Equals(temperatureInfos[ctr2]));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    (2, 97.9, 97.8, 98, 98.2) = (1, 98.6, 98.8, 98.8, 99): False
//    (2, 97.9, 97.8, 98, 98.2) = (2, 98.6, 98.6, 98.6, 98.4): False
//    (2, 97.9, 97.8, 98, 98.2) = (1, 98.4, 98.6, 99, 99.2): False
//    (2, 97.9, 97.8, 98, 98.2) = (2, 98.6, 98.6, 98.6, 98.4): False
//    (2, 97.9, 97.8, 98, 98.2) = (1, 98.6, 98.8, 98.8, 99): False
//
//    (1, 98.6, 98.8, 98.8, 99) = (2, 98.6, 98.6, 98.6, 98.4): False
//    (1, 98.6, 98.8, 98.8, 99) = (1, 98.4, 98.6, 99, 99.2): False
//    (1, 98.6, 98.8, 98.8, 99) = (2, 98.6, 98.6, 98.6, 98.4): False
//    (1, 98.6, 98.8, 98.8, 99) = (1, 98.6, 98.8, 98.8, 99): True
//
//    (2, 98.6, 98.6, 98.6, 98.4) = (1, 98.4, 98.6, 99, 99.2): False
//    (2, 98.6, 98.6, 98.6, 98.4) = (2, 98.6, 98.6, 98.6, 98.4): True
//    (2, 98.6, 98.6, 98.6, 98.4) = (1, 98.6, 98.8, 98.8, 99): False
//
//    (1, 98.4, 98.6, 99, 99.2) = (2, 98.6, 98.6, 98.6, 98.4): False
//    (1, 98.4, 98.6, 99, 99.2) = (1, 98.6, 98.8, 98.8, 99): False
//
//    (2, 98.6, 98.6, 98.6, 98.4) = (1, 98.6, 98.8, 98.8, 99): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2,T3,T4,T5> object.
- Its five components are of the same types as the current instance.
- Its five components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2,T3,T4,T5> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3,T4,T5> object.

Examples

The following example illustrates the ToString method. It displays an array of 5-tuple objects that contain the name of a state in the United States, its population in 1990 and 2000, its population change in this 10-year period, and the annual rate of population change.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Define array of tuples reflecting population change by state, 1990-
2000.
      Tuple<string, int, int, int, double>[] populationChanges =
           { Tuple.Create("California", 29760021, 33871648, 4111627, 13.8),
             Tuple.Create("Illinois", 11430602, 12419293, 988691, 8.6),
             Tuple.Create("Washington", 4866692, 5894121, 1027429, 21.1) };
      // Display each tuple.
      foreach (var item in populationChanges)
         Console.WriteLine(item.ToString());
```

```
    }
}
// The example displays the following output:
//       (California, 29760021, 33871648, 4111627, 13.8)
//       (Illinois, 11430602, 12419293, 988691, 8.6)
//       (Washington, 4866692, 5894121, 1027429, 21.1)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, `Item5`), where *Item1*, *Item2*, *Item3*, *Item4*, and `Item5` represent the values of the Item1, Item2, Item3, Item4, and Item5 properties, respectively. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

An object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3,T4,T5> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5> objects that contain career statistical data for running backs in American professional football. The 5-tuple's components consist of the player's name, the number of games in which he played, the number of carries or attempts, the total number of yards gained, and the number of touchdowns scored. The example displays the components of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display each tuple in sorted order. To sort the array, the example defines a generic `YardsGained` class that implements the IComparer interface and sorts the Tuple<T1,T2,T3,T4,T5> objects in descending order by the value of their fourth component (yards gained) rather than by their first component. Note that the example does not directly call the IStructuralComparable.CompareTo method. This method is called implicitly by the Array.Sort(Array, IComparer) method for each element in the array.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class YardsGained<T1, T2, T3, T4, T5> : IComparer
{
   public int Compare(object x, object y)
   {
      Tuple<T1, T2, T3, T4, T5> tX = x as Tuple<T1, T2, T3, T4, T5>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
```

```
         Tuple<T1, T2, T3, T4, T5> tY = y as Tuple<T1, T2, T3, T4, T5>;
         return -1 * Comparer<T4>.Default.Compare(tX.Item4, tY.Item4);
      }
   }
}

public class Example
{
   public static void Main()
   {
      // Organization of runningBacks 5-tuple:
      //    Component 1: Player name
      //    Component 2: Number of games played
      //    Component 3: Number of attempts (carries)
      //    Component 4: Number of yards gained
      //    Component 5: Number of touchdowns
      Tuple<string, int, int, int, int>[] runningBacks =
           { Tuple.Create("Payton, Walter", 190, 3838, 16726, 110),
             Tuple.Create("Sanders, Barry", 153, 3062, 15269, 99),
             Tuple.Create("Brown, Jim", 118, 2359, 12312, 106),
             Tuple.Create("Dickerson, Eric", 144, 2996, 13259, 90),
             Tuple.Create("Faulk, Marshall", 176, 2836, 12279, 100) };

      // Display the array in unsorted order.
      Console.WriteLine("The values in unsorted order:");
      foreach (var runningBack in runningBacks)
         Console.WriteLine(runningBack.ToString());
      Console.WriteLine();

      // Sort the array
      Array.Sort(runningBacks, new YardsGained<string, int, int, int, int>
());

      // Display the array in sorted order.
      Console.WriteLine("The values in sorted order:");
      foreach (var runningBack in runningBacks)
         Console.WriteLine(runningBack.ToString());
   }
}
// The example displays the following output:
//       The values in unsorted order:
//       (Payton, Walter, 190, 3838, 16726, 110)
//       (Sanders, Barry, 153, 3062, 15269, 99)
//       (Brown, Jim, 118, 2359, 12312, 106)
//       (Dickerson, Eric, 144, 2996, 13259, 90)
//       (Faulk, Marshall, 176, 2836, 12279, 100)
//
//       The values in sorted order:
//       (Brown, Jim, 118, 2359, 12312, 106)
//       (Dickerson, Eric, 144, 2996, 13259, 90)
//       (Faulk, Marshall, 176, 2836, 12279, 100)
//       (Payton, Walter, 190, 3838, 16726, 110)
//       (Sanders, Barry, 153, 3062, 15269, 99)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5> instance is cast to an IStructuralComparable interface.

Although this method can be called directly, it is most commonly called by collection-sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals(Object, IEqualityComparer)** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines a `DoubleComparer` class that implements the IEqualityComparer interface. The example instantiates two Tuple<T1,T2,T3,T4,T5>

objects by using a random number generator to populate their second through fifth components, casts the first instance to an IStructuralEquatable interface, and then uses a `DoubleComparer` object to test the two Tuple<T1,T2,T3,T4,T5> objects for approximate equality.

C#

```
using System;
using System.Collections;

public class DoubleComparer<T1, T2, T3, T4, T5> : IEqualityComparer
{
   private double difference;
   private int argument = 0;

   public DoubleComparer(double difference)
   {
      this.difference = difference;
   }

   new public bool Equals(object x, object y)
   {
      argument += 1;

      // Return true for Item1.
      if (argument == 1) return true;

      double d1 = (double) x;
      double d2 = (double) y;

      if (d1 - d2 < d1 * difference)
         return true;
      else
         return false;
   }

   public int GetHashCode(object obj)
   {
      if (obj is T1)
         return ((T1) obj).GetHashCode();
      else if (obj is T2)
         return ((T2) obj).GetHashCode();
      else if (obj is T3)
         return ((T3) obj).GetHashCode();
      else if (obj is T4)
         return ((T4) obj).GetHashCode();
      else
         return ((T5) obj).GetHashCode();
   }
}

public class Example
{
```

```
    public static void Main()
    {
       var value1 = GetValues(1);
       var value2 = GetValues(2);
       IStructuralEquatable iValue1 = value1;
       Console.WriteLine("{0} =\n{1} :\n{2}", value1, value2,
                         iValue1.Equals(value2,
                         new DoubleComparer<int, double, double, double,
double>(.01)));
    }

    private static Tuple<int, double, double, double, double> GetValues(int
ctr)
    {
       // Generate four random numbers between 0 and 1
       Random rnd = new Random((int)DateTime.Now.Ticks >> 32 >> ctr);
       return Tuple.Create(ctr, rnd.NextDouble(), rnd.NextDouble(),
                           rnd.NextDouble(), rnd.NextDouble());
    }
}
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3,T4,T5> object whose components are of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the Tuple<T1,T2,T3,T4,T5> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2,T3,T4,T5> objects. This continues until the method call returns `false` when it compares a specific pair of `Item` values, or the two Item5 values are passed to the method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.IStructural Equatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer
comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3,T4,T5> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3,T4,T5> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3,T4,T5> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5> objects that contain career statistics for running backs in American professional football. The five components consist of the player's name, the number of games in which he played, the number of carries or attempts, the total number of yards gained, and the number of touchdowns scored. The example displays the components of each tuple in the array in unsorted order, sorts the array, and then calls ToString to display each tuple in sorted order. The output shows that the array has been sorted by name, which is the first component. Note that the example does not directly call the IComparable.CompareTo method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      // Organization of runningBacks 5-tuple:
      //    Component 1: Player name
      //    Component 2: Number of games played
      //    Component 3: Number of attempts (carries)
      //    Component 4: Number of yards gained
      //    Component 5: Number of touchdowns
      Tuple<string, int, int, int, int>[] runningBacks =
           { Tuple.Create("Payton, Walter", 190, 3838, 16726, 110),
             Tuple.Create("Sanders, Barry", 153, 3062, 15269, 99),
             Tuple.Create("Brown, Jim", 118, 2359, 12312, 106),
             Tuple.Create("Dickerson, Eric", 144, 2996, 13259, 90),
             Tuple.Create("Faulk, Marshall", 176, 2836, 12279, 100) };

      // Display the array in unsorted order.
      Console.WriteLine("The values in unsorted order:");
      foreach (var runningBack in runningBacks)
         Console.WriteLine(runningBack.ToString());
      Console.WriteLine();
```

```
      // Sort the array
      Array.Sort(runningBacks);

      // Display the array in sorted order.
      Console.WriteLine("The values in sorted order:");
      foreach (var runningBack in runningBacks)
         Console.WriteLine(runningBack.ToString());
   }
}
// The example displays the following output:
//       The values in unsorted order:
//       (Payton, Walter, 190, 3838, 16726, 110)
//       (Sanders, Barry, 153, 3062, 15269, 99)
//       (Brown, Jim, 118, 2359, 12312, 106)
//       (Dickerson, Eric, 144, 2996, 13259, 90)
//       (Faulk, Marshall, 176, 2836, 12279, 100)
//
//       The values in sorted order:
//       (Brown, Jim, 118, 2359, 12312, 106)
//       (Dickerson, Eric, 144, 2996, 13259, 90)
//       (Faulk, Marshall, 176, 2836, 12279, 100)
//       (Payton, Walter, 190, 3838, 16726, 110)
//       (Sanders, Barry, 153, 3062, 15269, 99)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3,T4,T5> class. Although the method can be called directly, it is most commonly called by the default overloads of collection-sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Tuple<T1,T2,T3,T4,T5>.Equals(Object)** method.

The IComparable.CompareTo(Object) method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 4.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 4.

Remarks

The Tuple<T1,T2,T3,T4,T5>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5> object to an ITuple

interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

5, the number of elements in a Tuple<T1,T2,T3,T4,T5> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 6-tuple, or sextuple.

C#

```
public class Tuple<T1,T2,T3,T4,T5,T6> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

T4

The type of the tuple's fourth component.

T5

The type of the tuple's fifth component.

T6

The type of the tuple's sixth component.

Inheritance Object → Tuple<T1,T2,T3,T4,T5,T6>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3,T4,T5,T6> class represents a 6-tuple, or sextuple, which is a tuple that has six components.

You can instantiate a Tuple<T1,T2,T3,T4,T5,T6> object by calling either the Tuple<T1,T2,T3,T4,T5,T6> or the static Tuple.Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6) method. You can retrieve the value of the tuple's components by using the read-only Item1, Item2, Item3, Item4, Item5, and Item6 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set. The following example defines a Tuple<T1,T2,T3,T4,T5,T6> object that contains population data for New York City for each census from 1960 through 2000. The sextuple is passed to the `ComputePopulationChange` method, which calculates the annual rate of change between censuses, as well as the annual rate of change for the entire 50 year period.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get population data for New York City, 1960-2000.
      var population =
           Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
8008278);
      var rate = ComputePopulationChange(population);
      // Display results.
      Console.WriteLine("Population Change, {0}, 1960-2000\n",
population.Item1);
      Console.WriteLine("Year      {0,10} {1,9}", "Population", "Annual
Rate");
      Console.WriteLine("1960      {0,10:N0} {1,11}", population.Item2,
"NA");
      Console.WriteLine("1970      {0,10:N0} {1,11:P2}",
population.Item3, rate.Item2/10);
```

```
      Console.WriteLine("1980      {0,10:N0} {1,11:P2}",
population.Item4, rate.Item3/10);
      Console.WriteLine("1990      {0,10:N0} {1,11:P2}",
population.Item5, rate.Item4/10);
      Console.WriteLine("2000      {0,10:N0} {1,11:P2}",
population.Item6, rate.Item5/10);
      Console.WriteLine("1960-2000 {0,10:N0} {1,11:P2}", "",
rate.Item6/50);
   }

   private static Tuple<string, double, double, double, double, double>
ComputePopulationChange(
           Tuple<string, int, int, int, int, int> data)
   {
      var rate = Tuple.Create(data.Item1,
                              (double)(data.Item3 -
data.Item2)/data.Item2,
                              (double)(data.Item4 -
data.Item3)/data.Item3,
                              (double)(data.Item5 -
data.Item4)/data.Item4,
                              (double)(data.Item6 -
data.Item5)/data.Item5,
                              (double)(data.Item6 -
data.Item2)/data.Item2 );
      return rate;
   }
}
// The example displays the following output:
//       Population Change, New York, 1960-2000
//
//       Year      Population Annual Rate
//       1960       7,781,984          NA
//       1970       7,894,862      0.15 %
//       1980       7,071,639     -1.04 %
//       1990       7,322,564      0.35 %
//       2000       8,008,278      0.94 %
//       1960-2000                 0.06 %
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its computed statistics, along with the city name, in a Tuple<T1,T2,T3,T4,T5,T6> object.

- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2,T3,T4,T5,T6> object as the method argument, you can supply the thread's startup routine with six items of data.

Constructors

Expand table

Tuple<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's third component.
Item4	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fourth component.
Item5	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fifth component.
Item6	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's sixth component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2,T3,T4,T5,T6> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current Tuple<T1,T2,T3,T4,T5,T6> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3,T4,T5,T6> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>, T1, T2, T3, T4, T5, T6)	Deconstructs a tuple with 6 elements into separate variables.
ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6,T7>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

item4 T4

The value of the tuple's fourth component.

item5 T5

The value of the tuple's fifth component.

item6 T6

The value of the tuple's sixth component.

Remarks

You can also use the static Tuple.Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6) method to instantiate a 6-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6) method to instantiate a 6-tuple whose first component is of type String and whose remaining components are of type Int32.

C#

```
var tuple6 = Tuple.Create("Jane", 90, 87, 93, 67, 100);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}",
                  tuple6.Item1, tuple6.Item2, tuple6.Item3,
                  tuple6.Item4, tuple6.Item5, tuple6.Item6);
// Displays Test scores for Jane: 90, 87, 93, 67, 100
```

This is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5,T6> class constructor.

C#

```
var tuple6 = new Tuple<string, int, int, int, int, int>
                     ("Jane", 90, 87, 93, 67, 100);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}",
                  tuple6.Item1, tuple6.Item2, tuple6.Item3,
                  tuple6.Item4, tuple6.Item5, tuple6.Item6);
// Displays Test scores for Jane: 90, 87, 93, 67, 100
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's first component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the city name, it retrieves the value of the Item1 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's second component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the population in 1960, it retrieves the value of the Item2 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's third component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the population in 1970, it retrieves the value of the Item3 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item3 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item3 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item4 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fourth component.

C#

```
public T4 Item4 { get; }
```

Property Value

T4

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fourth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the population in 1980, it retrieves the value of the Item4 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item4 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item4 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item5 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fifth component.

C#

```
public T5 Item5 { get; }
```

Property Value

T5

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's fifth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the population in 1990, it retrieves the value of the Item5 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item5 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item5 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Item6 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6> object's sixth component.

C#

```
public T6 Item6 { get; }
```

Property Value

T6

The value of the current Tuple<T1,T2,T3,T4,T5,T6> object's sixth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1960 through 2000. It then displays a table that lists the data. To display the population in 2000, it retrieves the value of the Item6 property for each Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
```

```
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,60}",
                        "City", new String('-',(60-header.Length)/2) +
header +
                                new String('-', (60-header.Length)/2));
      Console.WriteLine("{0,25}{1,12}{2,12}{3,12}{4,12}\n",
                        "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,12:N0}{2,12:N0}{3,12:N0}{4,12:N0}
{5,12:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6);
   }
}
// The example displays the following output:
//    City          ----------------------Population in--------------------
---
//                         1960        1970        1980        1990
2000
//
//    New York        7,781,984   7,894,862   7,071,639   7,322,564
8,008,278
//    Los Angeles     2,479,015   2,816,061   2,966,850   3,485,398
3,694,820
//    Chicago         3,550,904   3,366,957   3,005,072   2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item6 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item6 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6> object is equal to a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example defines an array of sextuples that contain population data for Los Angeles and New York from 1960 to 2000. The first component of each sextuple identifies the city. The first, third, and fourth sextuples contain data for New York. The first sextuple is a duplicate of the fourth sextuple. The third sextuple identifies the city as "New York City" instead of "New York". As the example shows, only the fourth sextuple is equal to the first sextuple.

```
C#

using System;
```

```
public class Example
{
   public static void Main()
   {
      // Get population data for New York City and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int>[] urbanPopulations =
           { Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
8008278),
             Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
             Tuple.Create("New York City", 7781984, 7894862, 7071639,
7322564, 8008278),
             Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
8008278) };
      // Compare each tuple with every other tuple for equality.
      for (int ctr = 0; ctr <= urbanPopulations.Length - 2; ctr++)
      {
         var urbanPopulation = urbanPopulations[ctr];
         Console.WriteLine(urbanPopulation.ToString() + " = ");
         for (int innerCtr = ctr +1; innerCtr <= urbanPopulations.Length -
1; innerCtr++)
            Console.WriteLine("   {0}: {1}", urbanPopulations[innerCtr],

urbanPopulation.Equals(urbanPopulations[innerCtr]));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278) =
//       (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820): False
//       (New York City, 7781984, 7894862, 7071639, 7322564, 8008278): False
//       (New York, 7781984, 7894862, 7071639, 7322564, 8008278): True
//
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820) =
//       (New York City, 7781984, 7894862, 7071639, 7322564, 8008278): False
//       (New York, 7781984, 7894862, 7071639, 7322564, 8008278): False
//
//    (New York City, 7781984, 7894862, 7071639, 7322564, 8008278) =
//       (New York, 7781984, 7894862, 7071639, 7322564, 8008278): False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2,T3,T4,T5,T6> object.
- Its six components are of the same types as the current instance.

- Its six components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2,T3,T4,T5,T6> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3,T4,T5,T6> object.

Examples

The following example defines a Tuple<T1,T2,T3,T4,T5,T6> object that contains population data for New York City for each census from 1960 through 2000. The components of the sextuple are then displayed by a call to the ToString method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get population data for New York City, 1960-2000.
      var population = Tuple.Create("New York", 7781984, 7894862,
                                    7071639, 7322564, 8008278);
      Console.WriteLine(population.ToString());
   }
}
```

```
// The example displays the following output:
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, `Item5`, `Item6`), where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, and *Item6* represent the values of the Item1, Item2, Item3, Item4, Item5, and Item6 properties, respectively. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

An object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3,T4,T5,T6> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5,T6> objects that contains population data for three U.S. cities from 1960 to 2000. The sextuple's first component is the city name. The remaining five components represent the population at 10-year intervals from 1960 to 2000.

The `PopulationComparer` class provides an IComparer implementation that allows the array of sextuples to be sorted by any one of its components. Two values are provided to the `PopulationComparer` class in its constructor: The position of the component that defines the sort order, and a Boolean value that indicates whether the tuple objects should be sorted in ascending or descending order.

The example then displays the elements in the array in unsorted order, sorts them by the third component (the population in 1970) and displays them, and then sorts them by the sixth component (the population in 2000) and displays them.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class PopulationComparer<T1, T2, T3, T4, T5, T6> : IComparer
{
   private int itemPosition;
   private int multiplier = -1;

   public PopulationComparer(int component) : this(component, true)
   { }

   public PopulationComparer(int component, bool descending)
```

```
   {
      if (! descending) multiplier = 1;

      if (component <= 0 || component > 6)
         throw new ArgumentException("The component argument is out of
range.");

      itemPosition = component;
   }

   public int Compare(object x, object y)
   {
      var tX = x as Tuple<T1, T2, T3, T4, T5, T6>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
         var tY = y as Tuple<T1, T2, T3, T4, T5, T6>;
         switch (itemPosition)
         {
            case 1:
               return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
            case 2:
               return Comparer<T2>.Default.Compare(tX.Item2, tY.Item2) *
multiplier;
            case 3:
               return Comparer<T3>.Default.Compare(tX.Item3, tY.Item3) *
multiplier;
            case 4:
               return Comparer<T4>.Default.Compare(tX.Item4, tY.Item4) *
multiplier;
            case 5:
               return Comparer<T5>.Default.Compare(tX.Item5, tY.Item5) *
multiplier;
            case 6:
               return Comparer<T6>.Default.Compare(tX.Item6, tY.Item6) *
multiplier;
            default:
               return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
         }
      }
   }
}

public class Example
{
   public static void Main()
   {
      // Create array of sextuple with population data for three U.S.
      // cities, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
```

```
            { Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
              Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
8008278),
              Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display array in unsorted order.
      Console.WriteLine("In unsorted order:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int>(3));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 1970:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int>(6));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 2000:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
   }
}
// The example displays the following output:
//    In unsorted order:
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3550904, 3366957, 3005072, 2783726, 2896016)
//
//    Sorted by population in 1970:
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3550904, 3366957, 3005072, 2783726, 2896016)
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820)
//
//    Sorted by population in 2000:
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (Chicago, 3550904, 3366957, 3005072, 2783726, 2896016)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6> instance is cast to an IStructuralComparable interface.

This method lets you define customized comparisons of Tuple<T1,T2,T3,T4,T5,T6> objects. For example, you can use this method to order Tuple<T1,T2,T3,T4,T5,T6> objects based on the value of a specific component.

Although this method can be called directly, it is most commonly called by collection-sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals(Object, IEqualityComparer)** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines a `RateComparer` class that performs a custom test for equality. If the values that are passed to its IStructuralEquatable.Equals method are of

type Single or Double, the method tests for equality by using only up to three fractional digits. Otherwise, it simply calls the `x` parameter's `Equals` method. The example uses this IEqualityComparer implementation to determine whether four Tuple<T1,T2,T3,T4,T5,T6> objects that contain data on the rate of population change between 1960 and 2000 are equal.

C#

```
using System;
using System.Collections;

public class RateComparer<T1, T2, T3, T4, T5, T6> : IEqualityComparer
{
   private int argument = 0;

   public new bool Equals(object x, object y)
   {
      argument++;
      if (argument == 1) return true;

      double fx, fy;
      if (x is Double || x is Single)
      {
            fx = (double) x;
            fy = (double) y;
            return Math.Round(fx * 1000).Equals(Math.Round(fy * 1000));
      }
      else
      {
         return x.Equals(y);
      }
   }

   public int GetHashCode(object obj)
   {
      if (obj is Single || obj is Double)
         return Math.Round(((double) obj) * 1000).GetHashCode();
      else
         return obj.GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      var rate1 = Tuple.Create("New York", .014505, -.1042733,
                               .0354833, .093644, .0290792);
      var rate2 = Tuple.Create("Unknown City", .014505, -.1042733,
                               .0354833, .093644, .0290792);
      var rate3 = Tuple.Create("Unknown City", .014505, -.1042733,
                               .0354833, .093644, .029079);
```

```
        var rate4 = Tuple.Create("San Francisco", -.0332858, -.0512803,
                                 .0662544, .0728964, .0491912);
        IStructuralEquatable eq = rate1;
        // Compare first tuple with remaining two tuples.
        Console.WriteLine("{0} = ", rate1.ToString());
        Console.WriteLine("   {0} : {1}", rate2,
                          eq.Equals(rate2, new RateComparer<string, double,
double, double, double, double>()));
        Console.WriteLine("   {0} : {1}", rate3,
                          eq.Equals(rate3, new RateComparer<string, double,
double, double, double, double>()));
        Console.WriteLine("   {0} : {1}", rate4,
                          eq.Equals(rate4, new RateComparer<string, double,
double, double, double, double>()));
    }
}
// The example displays the following output:
//    (New York, 0.014505, -0.1042733, 0.0354833, 0.093644, 0.0290792) =
//       (Unknown City, 0.014505, -0.1042733, 0.0354833, 0.093644,
0.0290792) : True
//       (Unknown City, 0.014505, -0.1042733, 0.0354833, 0.093644, 0.029079)
: True
//       (San Francisco, -0.0332858, -0.0512803, 0.0662544, 0.0728964,
0.0491912) : False
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3,T4,T5,T6> object whose components are of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the Tuple<T1,T2,T3,T4,T5,T6> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2,T3,T4,T5,T6> objects. This continues until the method call returns `false` when it compares a specific pair of `Item` values, or the two Item6 values are passed to the method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.IStructural Equatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer
comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3,T4,T5,T6> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3,T4,T5,T6> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

```
C#

int IComparable.CompareTo (object obj);
```

Parameters

`obj` Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3,T4,T5,T6> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5,T6> objects that contain population data for three cities in the United States from 1960 to 2000. The six components consist of the city name followed by the city's population at 10-year intervals from 1960 to 2000. The example displays the components of each tuple in the array in unsorted order, sorts the array, and then calls the ToString method to display each tuple in sorted order. The output shows that the array has been sorted by name, which is the first component. Note that the example does not directly call the IComparable.CompareTo(Object) method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Create array of sextuple with population data for three U.S.
      // cities, 1960-2000.
      Tuple<string, int, int, int, int, int>[] cities =
          { Tuple.Create("Los Angeles", 2479015, 2816061, 2966850, 3485398,
3694820),
            Tuple.Create("New York", 7781984, 7894862, 7071639, 7322564,
8008278),
            Tuple.Create("Chicago", 3550904, 3366957, 3005072, 2783726,
2896016) };

      // Display array in unsorted order.
      Console.WriteLine("In unsorted order:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());

      Console.WriteLine();

      Array.Sort(cities);

      // Display array in sorted order.
```

```
        Console.WriteLine("In sorted order:");
        foreach (var city in cities)
           Console.WriteLine(city.ToString());
    }
}
// The example displays the following output:
//    In unsorted order:
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3550904, 3366957, 3005072, 2783726, 2896016)
//
//    In sorted order:
//    (Chicago, 3550904, 3366957, 3005072, 2783726, 2896016)
//    (Los Angeles, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7781984, 7894862, 7071639, 7322564, 8008278)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3,T4,T5,T6> class. Although the method can be called directly, it is most commonly called by the default overloads of collection-sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Tuple<T1,T2,T3,T4,T5,T6>.Equals(Object)** method.

The IComparable.CompareTo(Object) method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 5.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 5.

Remarks

The Tuple<T1,T2,T3,T4,T5,T6>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6> object

to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

6, the number of elements in a Tuple<T1,T2,T3,T4,T5,T6> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a 7-tuple, or septuple.

C#

```
public class Tuple<T1,T2,T3,T4,T5,T6,T7> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

T4

The type of the tuple's fourth component.

T5

The type of the tuple's fifth component.

T6

The type of the tuple's sixth component.

T7

The type of the tuple's seventh component.

Inheritance Object → Tuple<T1,T2,T3,T4,T5,T6,T7>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3,T4,T5,T6,T7> class represents a 7-tuple, or septuple, which is a tuple that has seven components.

You can instantiate a Tuple<T1,T2,T3,T4,T5,T6,T7> object by calling either the Tuple<T1,T2,T3,T4,T5,T6,T7> or the static Tuple.Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) method. You can retrieve the value of the tuple's components by using the read-only Item1, Item2, Item3, Item4, Item5, Item6, and Item7 instance properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set. The following example defines a Tuple<T1,T2,T3,T4,T5,T6,T7> object that contains population data for New York City for each census from 1950 through 2000. The septuple is passed to the `ComputePopulationChange` method, which calculates the annual rate of change between censuses, as well as the annual rate of change for the entire 60 year period.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get population data for New York City, 1950-2000.
      var population = Tuple.Create("New York", 7891957, 7781984,
                                    7894862, 7071639, 7322564,
8008278);
      var rate = ComputePopulationChange(population);
      // Display results.
      Console.WriteLine("Population Change, {0}, 1950-2000\n",
population.Item1);
      Console.WriteLine("Year      {0,10} {1,9}", "Population", "Annual
Rate");
      Console.WriteLine("1950      {0,10:N0} {1,11}", population.Item2,
```

```
"NA");
      Console.WriteLine("1960      {0,10:N0} {1,11:P2}",
population.Item3, rate.Item2/10);
      Console.WriteLine("1970      {0,10:N0} {1,11:P2}",
population.Item4, rate.Item3/10);
      Console.WriteLine("1980      {0,10:N0} {1,11:P2}",
population.Item5, rate.Item4/10);
      Console.WriteLine("1990      {0,10:N0} {1,11:P2}",
population.Item6, rate.Item5/10);
      Console.WriteLine("2000      {0,10:N0} {1,11:P2}",
population.Item7, rate.Item6/10);
      Console.WriteLine("1950-2000 {0,10:N0} {1,11:P2}", "",
rate.Item7/50);
   }

   private static Tuple<string, double, double, double, double, double,
double>
        ComputePopulationChange(
           Tuple<string, int, int, int, int, int, int> data)
   {
      var rate = Tuple.Create(data.Item1,
                        (double)(data.Item3 - data.Item2)/data.Item2,
                        (double)(data.Item4 - data.Item3)/data.Item3,
                        (double)(data.Item5 - data.Item4)/data.Item4,
                        (double)(data.Item6 - data.Item5)/data.Item5,
                        (double)(data.Item7 - data.Item6)/data.Item6,
                        (double)(data.Item7 - data.Item2)/data.Item2 );
      return rate;
   }
}
// The example displays the following output:
//       Population Change, New York, 1950-2000
//
//       Year      Population Annual Rate
//       1950       7,891,957          NA
//       1960       7,781,984     -0.14 %
//       1970       7,894,862      0.15 %
//       1980       7,071,639     -1.04 %
//       1990       7,322,564      0.35 %
//       2000       8,008,278      0.94 %
//       1950-2000                 0.03 %
```

- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its computed statistics, along with the city name, in a Tuple<T1,T2,T3,T4,T5,T6,T7> object.

- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a

Tuple<T1,T2,T3,T4,T5,T6,T7> object as the method argument, you can supply the thread's startup routine with seven items of data.

Constructors

Expand table

Tuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6,T7> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's third component.
Item4	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fourth component.
Item5	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fifth component.
Item6	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's sixth component.
Item7	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's seventh component.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7> object is equal to a specified object.
GetHashCode()	Returns the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6,T7>

	instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.
ITuple.Length	Gets the number of elements in the `Tuple`.

Extension Methods

Expand table

Deconstruct<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>, T1, T2, T3, T4, T5, T6, T7)	Deconstructs a tuple with 7 elements into separate variables.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6,T7> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6, T7
item7);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

item4 T4

The value of the tuple's fourth component.

item5 T5

The value of the tuple's fifth component.

item6 T6

The value of the tuple's sixth component.

item7 T7

The value of the tuple's seventh component.

Remarks

You can use the static Tuple.Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) method to instantiate a 7-tuple object without having to explicitly specify the types of its components. The following example uses the Tuple.Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) method to instantiate a 7-tuple whose first component is of type String and whose remaining components are of type Int32.

C#

```
var tuple7 = Tuple.Create("Jane", 90, 87, 93, 67, 100, 92);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}, {6}",
                  tuple7.Item1, tuple7.Item2, tuple7.Item3,
                  tuple7.Item4, tuple7.Item5, tuple7.Item6,
                  tuple7.Item7);
// Displays Test scores for Jane: 90, 87, 93, 67, 100, 92
```

This is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5,T6,T7> class constructor.

C#

```
var tuple7 = new Tuple<string, int, int, int, int, int, int>
                      ("Jane", 90, 87, 93, 67, 100, 92);
Console.WriteLine("Test scores for {0}: {1}, {2}, {3}, {4}, {5}, {6}",
                  tuple7.Item1, tuple7.Item2, tuple7.Item3,
                  tuple7.Item4, tuple7.Item5, tuple7.Item6,
                  tuple7.Item7);
// Displays Test scores for Jane: 90, 87, 93, 67, 100, 92
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's first component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the city name, it retrieves the value of the Item1 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's second component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 1950, it retrieves the value of the Item2 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's third component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 1960, it retrieves the value of the Item3 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item3 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item3 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the third element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item4 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fourth component.

C#

```
public T4 Item4 { get; }
```

Property Value

T4

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fourth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 1970, it retrieves the value of the Item4 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item4 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item4 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the fourth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item5 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fifth component.

C#

```
public T5 Item5 { get; }
```

Property Value

T5

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's fifth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 1980, it retrieves the value of the Item5 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item5 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item5 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the fifth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item6 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's sixth component.

C#

```
public T6 Item6 { get; }
```

Property Value

T6

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's sixth component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 1990, it retrieves the value of the Item6 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item6 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item6 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the sixth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Item7 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's seventh component.

C#

```
public T7 Item7 { get; }
```

Property Value

T7

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object's seventh component.

Examples

The following example defines an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects whose components contain population data for three U.S. cities (New York, Chicago, and Los Angeles) from 1950 through 2000. It then displays a table that lists the data. To display the population in 2000, it retrieves the value of the Item7 property for each Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      // Create tuples containing population data for New York, Chicago,
      // and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
```

```
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display tuple data in table.
      string header = "Population in";
      Console.WriteLine("{0,-12} {1,66}",
                        "City", new String('-',(66-header.Length)/2) +
header +
                                new String('-', (66-header.Length)/2));
      Console.WriteLine("{0,24}{1,11}{2,11}{3,11}{4,11}{5,11}\n",
                        "1950", "1960", "1970", "1980", "1990", "2000");

      foreach (var city in cities)
         Console.WriteLine("{0,-12} {1,11:N0}{2,11:N0}{3,11:N0}{4,11:N0}
{5,11:N0}{6,11:N0}",
                           city.Item1, city.Item2, city.Item3, city.Item4,
                           city.Item5, city.Item6, city.Item7);
   }
}
// The example displays the following output:
//   City          -------------------------Population in-----------------
--------
//                       1950       1960       1970       1980       1990
2000
//
//   New York       7,891,957  7,781,984  7,894,862  7,071,639  7,322,564
8,008,278
//   Los Angeles    1,970,358  2,479,015  2,816,061  2,966,850  3,485,398
3,694,820
//   Chicago        3,620,962  3,550,904  3,366,957  3,005,072  2,783,726
2,896,016
```

Remarks

You can dynamically determine the type of the Item7 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item7 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7> object, and retrieving the seventh element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7> object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example defines an array of sextuples that contain population data for Los Angeles and New York from 1950 to 2000. The first component of each septuple identifies the city. The first, third, and fourth septuples contain data for New York. The first septuple is a duplicate of the fourth septuple. The third septuple identifies the city as "New York City" instead of "New York". As the example shows, only the fourth septuple is equal to the first septuple.

C#

```
using System;
```

```
public class Example
{
   public static void Main()
   {
      // Get population data for New York City and Los Angeles, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] urbanPopulations =
           { Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
7322564, 8008278),
             Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("New York City", 7891957, 7781984, 7894862,
7071639, 7322564, 8008278),
             Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
7322564, 8008278) };
      // Compare each tuple with every other tuple for equality.
      for (int ctr = 0; ctr <= urbanPopulations.Length - 2; ctr++)
      {
         var urbanPopulation = urbanPopulations[ctr];
         Console.WriteLine(urbanPopulation.ToString() + " = ");
         for (int innerCtr = ctr +1; innerCtr <= urbanPopulations.Length -
1; innerCtr++)
            Console.WriteLine("   {0}: {1}", urbanPopulations[innerCtr],

urbanPopulation.Equals(urbanPopulations[innerCtr]));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278) =
//       (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398,
3694820): False
//       (New York City, 7891957, 7781984, 7894862, 7071639, 7322564,
8008278): False
//       (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278):
True
//
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820) =
//       (New York City, 7891957, 7781984, 7894862, 7071639, 7322564,
8008278): False
//       (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278):
False
//
//    (New York City, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
=
//       (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278):
False
```

Remarks

The `obj` parameter is considered to be equal to the current instance under the following conditions:

- It is a Tuple<T1,T2,T3,T4,T5,T6,T7> object.
- Its seven components are of the same types as the current instance.
- Its seven components are equal to those of the current instance. Equality is determined by the default object equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6,T7> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3,T4,T5,T6,T7> object.

Examples

The following example defines a Tuple<T1,T2,T3,T4,T5,T6,T7> object that contains population data for New York City for each census from 1950 through 2000. The components of the sextuple are then displayed by a call to the ToString method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Get population data for New York City, 1960-2000.
      var population = Tuple.Create("New York", 7891957, 7781984,
                                    7894862, 7071639, 7322564, 8008278);
      Console.WriteLine(population.ToString());
   }
}
```

```
// The example displays the following output:
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, `Item5, Item6, Item7`), where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, and *Item7* represent the values of the Item1, Item2, Item3, Item4, Item5, Item6, and Item7 properties, respectively. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7> object to a specified object by using a specified comparer, and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

An object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3,T4,T5,T6,T7> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects that contains population data for three U.S. cities from 1950 to 2000. The septuple's first component is the city name. The remaining five components represent the population at 10-year intervals from 1950 to 2000.

The `PopulationComparer` class provides an IComparer implementation that allows the array of septuples to be sorted by any one of its components. Two values are provided to the `PopulationComparer` class in its constructor: The position of the component that defines the sort order, and a Boolean value that indicates whether the tuple objects should be sorted in ascending or descending order.

The example then displays the elements in the array in unsorted order, sorts them by the third component (the population in 1960) and displays them, and then sorts them by the sixth component (the population in 1990) and displays them.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class PopulationComparer<T1, T2, T3, T4, T5, T6, T7> : IComparer
{
   private int itemPosition;
   private int multiplier = -1;

   public PopulationComparer(int component) : this(component, true)
   { }

   public PopulationComparer(int component, bool descending)
```

```
   {
      if (! descending) multiplier = 1;

      if (component <= 0 || component > 7)
         throw new ArgumentException("The component argument is out of
range.");

      itemPosition = component;
   }

   public int Compare(object x, object y)
   {
      Tuple<T1, T2, T3, T4, T5, T6, T7> tX = x as Tuple<T1, T2, T3, T4, T5,
T6, T7>;
      if (tX == null)
      {
         return 0;
      }
      else
      {
         Tuple<T1, T2, T3, T4, T5, T6, T7> tY = y as Tuple<T1, T2, T3, T4,
T5, T6, T7>;
         switch (itemPosition)
         {
            case 1:
               return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
            case 2:
               return Comparer<T2>.Default.Compare(tX.Item2, tY.Item2) *
multiplier;
            case 3:
               return Comparer<T3>.Default.Compare(tX.Item3, tY.Item3) *
multiplier;
            case 4:
               return Comparer<T4>.Default.Compare(tX.Item4, tY.Item4) *
multiplier;
            case 5:
               return Comparer<T5>.Default.Compare(tX.Item5, tY.Item5) *
multiplier;
            case 6:
               return Comparer<T6>.Default.Compare(tX.Item6, tY.Item6) *
multiplier;
            case 7:
               return Comparer<T7>.Default.Compare(tX.Item7, tY.Item7) *
multiplier;
            default:
               return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
         }
      }
   }
}

public class Example
{
```

```
   public static void Main()
   {
      // Create array of sextuple with population data for three U.S.
      // cities, 1960-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
           { Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
             Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
7322564, 8008278),
             Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display array in unsorted order.
      Console.WriteLine("In unsorted order:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int, int>(3));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 1960:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int, int>(6));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 1990:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
   }
}
// The example displays the following output:
//    In unsorted order:
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3620962, 3550904, 3366957, 3005072, 2783726, 2896016)
//
//    Sorted by population in 1960:
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3620962, 3550904, 3366957, 3005072, 2783726, 2896016)
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820)
//
//    Sorted by population in 1990:
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (Chicago, 3620962, 3550904, 3366957, 3005072, 2783726, 2896016)
```

Remarks

This member is an explicit interface implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7> instance is cast to an IStructuralComparable interface.

This method lets you define customized comparisons of Tuple<T1,T2,T3,T4,T5,T6,T7> objects. For example, you can use this method to order Tuple<T1,T2,T3,T4,T5,T6,T7> objects based on the value of a specific component.

Although this method can be called directly, it is most commonly called by collection-sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IStructuralEquatable.Equals(Object, IEqualityComparer)

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Examples

The following example defines a `RateComparer` class that performs a custom test for equality. If the values that are passed to its IStructuralEquatable.Equals method are of

type Single or Double, the method tests for equality by using only up to three fractional digits. Otherwise, it simply calls the `x` parameter's `Equals` method. The example uses this IEqualityComparer implementation to determine whether four Tuple<T1,T2,T3,T4,T5,T6,T7> objects that contain data on the rate of population change between 1950 and 2000 are equal.

C#

```
using System;
using System.Collections;

public class RateComparer<T1, T2, T3, T4, T5, T6, T7> : IEqualityComparer
{
   private int argument = 0;

   public new bool Equals(object x, object y)
   {
      argument++;
      if (argument == 1) return true;

      double fx, fy;
      if (x is Double || x is Single)
      {
            fx = (double) x;
            fy = (double) y;
            return Math.Round(fx * 1000).Equals(Math.Round(fy * 1000));
      }
      else
      {
         return x.Equals(y);
      }
   }

   public int GetHashCode(object obj)
   {
      if (obj is Single || obj is Double)
         return Math.Round(((double) obj) * 1000).GetHashCode();
      else
         return obj.GetHashCode();
   }
}

public class Example
{
   public static void Main()
   {
      var rate1 = Tuple.Create("New York", -.013934, .014505,
                               -.1042733, .0354833, .093644, .0290792);
      var rate2 = Tuple.Create("Unknown City", -.013934, .014505,
                               -.1042733, .0354833, .093644, .0290792);
      var rate3 = Tuple.Create("Unknown City", -.013934, .014505,
                               -.1042733, .0354833, .093644, .029079);
```

```
        var rate4 = Tuple.Create("San Francisco", -.0451934, -.0332858,
                                 -.0512803, .0662544, .0728964, .0491912);
        IStructuralEquatable eq = rate1;
        // Compare first tuple with remaining two tuples.
        Console.WriteLine("{0} = ", rate1.ToString());
        Console.WriteLine("   {0} : {1}", rate2,
                          eq.Equals(rate2, new RateComparer<string, double,
double, double, double, double, double>()));
        Console.WriteLine("   {0} : {1}", rate3,
                          eq.Equals(rate3, new RateComparer<string, double,
double, double, double, double, double>()));
        Console.WriteLine("   {0} : {1}", rate4,
                          eq.Equals(rate4, new RateComparer<string, double,
double,double,  double, double, double>()));
    }
}
// The example displays the following output:
//    (New York, -0.013934, 0.014505, -0.1042733, 0.0354833, 0.093644,
0.0290792) =
//       (Unknown City, -0.013934, 0.014505, -0.1042733, 0.0354833,
0.093644, 0.0290792) : True
//       (Unknown City, -0.013934, 0.014505, -0.1042733, 0.0354833,
0.093644, 0.029079) : True
//       (San Francisco, -0.0451934, -0.0332858, -0.0512803, 0.0662544,
0.0728964, 0.0491912) : False
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if other is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3,T4,T5,T6,T7> object whose components are of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the Tuple<T1,T2,T3,T4,T5,T6,T7> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2,T3,T4,T5,T6,T7> objects. This continues until the method call returns `false` when it compares a specific pair of `Item` values, or the two Item7 values are passed to the method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3,T4,T5,T6,T7> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7> instance is cast to an IStructuralEquatable interface.

the method simply wraps a call to the comparer object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3,T4,T5,T6,T7> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

```
C#

int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3,T4,T5,T6,T7> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5,T6,T7> objects that contain population data for three cities in the United States from 1950 to 2000. The seven components consist of the city name followed by the city's population at 10-year intervals from 1950 to 2000. The example displays the components of each tuple in the array in unsorted order, sorts the array, and then calls the ToString method to display each tuple in sorted order. The output shows that the array has been sorted by name, which is the first component. Note that the example does not directly call the IComparable.CompareTo(Object) method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Create array of sextuple with population data for three U.S.
      // cities, 1950-2000.
      Tuple<string, int, int, int, int, int, int>[] cities =
          { Tuple.Create("Los Angeles", 1970358, 2479015, 2816061, 2966850,
3485398, 3694820),
            Tuple.Create("New York", 7891957, 7781984, 7894862, 7071639,
7322564, 8008278),
            Tuple.Create("Chicago", 3620962, 3550904, 3366957, 3005072,
2783726, 2896016) };

      // Display array in unsorted order.
      Console.WriteLine("In unsorted order:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());

      Console.WriteLine();

      Array.Sort(cities);

      // Display array in sorted order.
```

```
        Console.WriteLine("In sorted order:");
        foreach (var city in cities)
           Console.WriteLine(city.ToString());
    }
}
// The example displays the following output:
//    In unsorted order:
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
//    (Chicago, 3620962, 3550904, 3366957, 3005072, 2783726, 2896016)
//
//    In sorted order:
//    (Chicago, 3620962, 3550904, 3366957, 3005072, 2783726, 2896016)
//    (Los Angeles, 1970358, 2479015, 2816061, 2966850, 3485398, 3694820)
//    (New York, 7891957, 7781984, 7894862, 7071639, 7322564, 8008278)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3,T4,T5,T6,T7> class. Although the method can be called directly, it is most commonly called by the default overloads of collection-sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Tuple<T1,T2,T3,T4,T5,T6,T7>.Equals(Object)** method.

This method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 to 6.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 6.

Remarks

The Tuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6,T7>

object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

7, the number of elements in a Tuple<T1,T2,T3,T4,T5,T6,T7> object.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6,T7> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an *n*-tuple, where *n* is 8 or greater.

C#

```
public class Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> : IComparable,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the tuple's first component.

T2

The type of the tuple's second component.

T3

The type of the tuple's third component.

T4

The type of the tuple's fourth component.

T5

The type of the tuple's fifth component.

T6

The type of the tuple's sixth component.

T7

The type of the tuple's seventh component.

`TRest`

Any generic `Tuple` object that defines the types of the tuple's remaining components.

Inheritance Object → Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Implements IStructuralComparable , IStructuralEquatable , IComparable , ITuple

Remarks

A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class represents an *n*-tuple that has eight or more components.

You can instantiate a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object with exactly eight components by calling the static Tuple.Create method. The following example creates an 8-tuple (octuple) that contains prime numbers that are less than 20. Note that it uses type inference to determine the type of each component.

C#

```
var primes = Tuple.Create(2, 3, 5, 7, 11, 13, 17, 19);
Console.WriteLine("Prime numbers less than 20: " +
                  "{0}, {1}, {2}, {3}, {4}, {5}, {6}, and {7}",
                  primes.Item1, primes.Item2, primes.Item3,
                  primes.Item4, primes.Item5, primes.Item6,
                  primes.Item7, primes.Rest.Item1);
// The example displays the following output:
//    Prime numbers less than 20: 2, 3, 5, 7, 11, 13, 17, and 19
```

You can also instantiate an n-tuple object with eight or more components by calling the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> constructor. The following example uses the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> constructor to create an 8-tuple that is equivalent to the tuple created in the previous example.

C#

```
var primes = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32, Int32,
             Tuple<Int32>> (2, 3, 5, 7, 11, 13, 17, new Tuple<Int32>(19));
```

> **Note**
>
> To create an n-tuple with nine or more components, you must call the **Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>** constructor. The static factory methods of the

Tuple class do not support the creation of `Tuple` objects with more than eight components.

To instantiate an n-tuple that has eight or more components with the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> constructor, you supply a generic `Tuple` object as the `rest` parameter to define the eighth through *n*th components of the tuple. By nesting generic `Tuple` objects in this way, you can create a tuple that has no practical limitation on the number of its components.

The following example creates a 17-tuple that contains population data for the city of Detroit, Michigan, for each national census from 1860 to 2000. The first component of the tuple is the city name. The second component is the start date of the data series, and the third component is the population at the start date. Each subsequent component provides the population at decade intervals. The 17-tuple is created by nesting a Tuple<T1,T2,T3> object inside a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object. (That is, the Tuple<T1,T2,T3> object is supplied as the value of the `rest` parameter in the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class constructor.) This Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is, in turn, nested in an outer Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object. (That is, the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is supplied as the value of the `rest` parameter in the outer Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's class constructor.)

C#

```
var from1980 = Tuple.Create(1203339, 1027974, 951270);
var from1910 = new Tuple<int, int, int, int, int, int, int, Tuple<int, int,
int>>
    (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462, from1980);
var population = new Tuple<string, int, int, int, int, int, int,
    Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
    ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704, from1910);
```

You can retrieve the value of the tuple's first seven components by using the read-only Item1, Item2, Item3, Item4, Item5, Item6, and Item7 instance properties. Any additional components are nested and can be retrieved from the Rest property. In the previous example, the Item1 through Item7 properties retrieve the first through seventh components of the tuple. The eighth through fourteenth components are contained in the tuple that is nested at the second level, and are represented by the `Rest.Item1` through `Rest.Item7` properties. The fifteenth through seventeenth components are contained in the tuple that is nested at the third level, and are represented by the `Rest.Rest.Item1` though `Rest.Rest.Item3` properties.

Tuples are commonly used in four different ways:

- To represent a single set of data. For example, a tuple can represent a database record, and its components can represent individual fields of the record.
- To provide easy access to, and manipulation of, a data set.
- To return multiple values from a method without the use of `out` parameters (in C#) or `ByRef` parameters (in Visual Basic). For example, the previous example returns its computed statistics, along with the city name, in a Tuple<T1,T2,T3,T4,T5,T6,T7> object.
- To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1,T2,T3,T4,T5,T6,T7> object as the method argument, you can supply the thread's startup routine with seven items of data.

Constructors

Expand table

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>(T1, T2, T3, T4, T5, T6, T7, TRest)	Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class.

Properties

Expand table

Item1	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's first component.
Item2	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's second component.
Item3	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's third component.
Item4	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fourth component.
Item5	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fifth component.
Item6	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's sixth component.
Item7	Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's seventh

	component.
Rest	Gets the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's remaining components.

Methods

Expand table

Equals(Object)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is equal to a specified object.
GetHashCode()	Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Explicit Interface Implementations

Expand table

IComparable.Compare To(Object)	Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable. CompareTo(Object, IComparer)	Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable. Equals(Object, IEquality Comparer)	Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is equal to a specified object based on a specified comparison method.
IStructuralEquatable.Get HashCode(IEquality Comparer)	Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `Tuple` element.

ITuple.Length	Gets the number of elements in the `Tuple`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Tuple
- Tuple<T1>
- Tuple<T1,T2>
- Tuple<T1,T2,T3>
- Tuple<T1,T2,T3,T4>
- Tuple<T1,T2,T3,T4,T5>
- Tuple<T1,T2,T3,T4,T5,T6>
- Tuple<T1,T2,T3,T4,T5,T6,T7>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class.

C#

```
public Tuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6, T7
item7, TRest rest);
```

Parameters

item1 T1

The value of the tuple's first component.

item2 T2

The value of the tuple's second component.

item3 T3

The value of the tuple's third component.

item4 T4

The value of the tuple's fourth component.

item5 T5

The value of the tuple's fifth component.

item6 T6

The value of the tuple's sixth component.

item7 T7

The value of the tuple's seventh component.

`rest` TRest

Any generic `Tuple` object that contains the values of the tuple's remaining components.

Exceptions

ArgumentException

`rest` is not a generic `Tuple` object.

Examples

The following example creates a 17-tuple that contains population data for the city of Detroit, Michigan, for each census from 1860 to 2000. The first component of the tuple is the city name. The second component is the start date of the series of data, and the third component is the population at the start date. Each subsequent component provides the population at decade intervals. The example uses two layers of nesting to create the 17-tuple: It defines a 7-tuple whose third through seventh components contain population data for 1860 through 1900, a nested 7-tuple that contains population data for 1910 through 1970, and an inner nested 3-tuple that contains population data for 1980 through 2000.

C#

```
var from1980 = Tuple.Create(1203339, 1027974, 951270);
var from1910 = new Tuple<int, int, int, int, int, int, int, Tuple<int, int,
int>>
    (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462, from1980);
var population = new Tuple<string, int, int, int, int, int, int,
    Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
    ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704, from1910);
```

Remarks

You can also use the static Tuple.Create method to instantiate an 8-tuple (octuple) object without having to explicitly specify the types of its components. The following example uses the Tuple.Create method to instantiate an 8-tuple object that contains prime numbers that are less than 20.

C#

```
var primes = Tuple.Create(2, 3, 5, 7, 11, 13, 17, 19);
Console.WriteLine("Prime numbers less than 20: " +
                  "{0}, {1}, {2}, {3}, {4}, {5}, {6}, and {7}",
                  primes.Item1, primes.Item2, primes.Item3,
```

```
                    primes.Item4, primes.Item5, primes.Item6,
                    primes.Item7, primes.Rest.Item1);
// The example displays the following output:
//    Prime numbers less than 20: 2, 3, 5, 7, 11, 13, 17, and 19
```

This is equivalent to the following call to the Tuple<T1,T2,T3,T4,T5,T6,T7> class constructor.

```
C#

var primes = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32, Int32,
             Tuple<Int32>> (2, 3, 5, 7, 11, 13, 17, new Tuple<Int32>(19));
```

However, the static Tuple.Create method cannot be used to create a tuple object with more than eight components.

When using the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> constructor to create an n-tuple with eight or more components, you use the `rest` parameter to create a nested n-tuple that has from one to seven components. By using successive levels of nesting, you can create an n-tuple that has a virtually unlimited number of components. For example, to create a 25-tuple, you instantiate a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object with three levels of nesting, as follows:

- The outermost Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object contains the first through seventh components. Its Rest property provides access to an Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object at the first level of nesting.
- The outermost nested Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object contains the eighth through fourteenth components, and its Rest property provides access to an Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object at the second level of nesting.
- The Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object at the second level of nesting contains the fifteenth through twenty-first components, and its Rest property provides access to an Tuple<T1,T2,T3,T4> object at the third level of nesting.
- The innermost tuple is a Tuple<T1,T2,T3,T4> object that contains the twenty-second through twenty-fifth components.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item1 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's first component.

C#

```
public T1 Item1 { get; }
```

Property Value

T1

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's first component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The first component of the 17-tuple is the city name. The example uses the Item1 property to display the city name in the table header before displaying the population data.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item1 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item1 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the first element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item2 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's second component.

C#

```
public T2 Item2 { get; }
```

Property Value

T2

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's second component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The second component of the 17-tuple is the first year of the data series. The example assigns the value of the Item2 property to a variable, and then uses that variable to represent the year whose data is displayed.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item2 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item2 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the second element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item3 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's third component.

C#

```
public T3 Item3 { get; }
```

Property Value

T3

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's third component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The third component of the 17-tuple is the population in 1860. The example uses the Item3 property to display the population value and to calculate the annual rate of population change between 1860 and 1870.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item3 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item3 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the third element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item4 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fourth component.

C#

```
public T4 Item4 { get; }
```

Property Value

T4

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fourth component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The fourth component of the 17-tuple is the population in 1870. The example uses the Item4 property to display the population value and to calculate the annual rate of population change between 1870 and 1880.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item4 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item4 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the fourth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item5 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fifth component.

C#

```
public T5 Item5 { get; }
```

Property Value

T5

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's fifth component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The fifth component of the 17-tuple is the population in 1880. The example uses the Item5 property to display the population value and to calculate the annual rate of population change between 1880 and 1890.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item5 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item5 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the fifth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item6 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's sixth component.

C#

```
public T6 Item6 { get; }
```

Property Value

T6

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's sixth component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The sixth component of the 17-tuple is the population in 1890. The example uses the Item6 property to display the population value and to calculate the annual rate of population change between 1890 and 1900.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item6 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item6 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the sixth element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item7 Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's seventh component.

C#

```
public T7 Item7 { get; }
```

Property Value

T7

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's seventh component.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The seventh component of the 17-tuple is the population in 1900. The example uses the Item7 property to display the population value and to calculate the annual rate of population change between 1900 and 1910.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
```

```
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
```

```
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

You can dynamically determine the type of the Item7 component in one of two ways:

- By calling the `GetType` method on the value that is returned by the Item7 property.
- By retrieving the Type object that represents the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object, and retrieving the seventh element from the array that is returned by its Type.GetGenericArguments method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's remaining components.

C#

```
public TRest Rest { get; }
```

Property Value

TRest

The value of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's remaining components.

Examples

The following example creates a 17-tuple object that contains population data for the city of Detroit, Michigan, from 1860 to 2000. The seventh component of the 17-tuple is the population in 1900. The example uses the Rest property to retrieve the values of the eighth through fourteenth components, and it uses the Rest property of the nested Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to retrieve the values of the remaining components.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
```

```
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
from1980);
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine("Population of {0}", population.Item1);
        Console.WriteLine();
        Console.WriteLine("{0,5}  {1,14}  {2,10}", "Year", "Population",
"Change");

        int year = population.Item2;
        ShowPopulation(year, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item4, population.Item3);
        year += 10;
        ShowPopulationChange(year, population.Item5, population.Item4);
        year += 10;
        ShowPopulationChange(year, population.Item6, population.Item5);
        year += 10;
        ShowPopulationChange(year, population.Item7, population.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item1, population.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item2,
population.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item3,
population.Rest.Item2);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item4,
population.Rest.Item3);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item5,
population.Rest.Item4);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item6,
population.Rest.Item5);
        year += 10;
        ShowPopulationChange(year, population.Rest.Item7,
population.Rest.Item6);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item1,
population.Rest.Item7);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item2,
population.Rest.Rest.Item1);
        year += 10;
        ShowPopulationChange(year, population.Rest.Rest.Item3,
population.Rest.Rest.Item2);
    }
```

```
    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//
//    Population of Detroit
//    Year      Population      Change
//    1860          45,619         n/a
//    1870          79,577      7.44 %
//    1880         116,340      4.62 %
//    1890         205,876      7.70 %
//    1900         285,704      3.88 %
//    1910         465,766      6.30 %
//    1920         993,078     11.32 %
//    1930       1,568,622      5.80 %
//    1940       1,623,452      0.35 %
//    1950       1,849,568      1.39 %
//    1960       1,670,144     -0.97 %
//    1970       1,511,462     -0.95 %
//    1980       1,203,339     -2.04 %
//    1990       1,027,974     -1.46 %
//    2000         951,270     -0.75 %
```

Remarks

The Rest property returns a nested `Tuple` object that allows access to the eighth though *n*th components of the tuple. Depending on the total number of components in the tuple, the values of the eighth through fourteenth components can be retrieved from the nested `Tuple` object's Item1 through Item7 properties. You can then use the Rest property of a nested Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to retrieve the `Tuple` object at the next level of nesting.

You can dynamically determine the number of components in a nested `Tuple` object that is returned by the Rest property by extracting the digit from its type name. The following example provides an illustration.

```
VB
```

```
Imports System.Reflection

Module Example
    Sub Main()
        Dim from1980 As Tuple(Of Integer, Integer, Integer) =
            Tuple.Create(1203339, 1027974, 951270)
        Dim from1910 As New Tuple(Of Integer, Integer, Integer, Integer,
Integer, Integer, Integer, _
            Tuple(Of Integer, Integer, Integer)) _
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
from1980)
        Dim population As New Tuple(Of String, Integer, Integer, Integer,
Integer, Integer, Integer, _
            Tuple(Of Integer, Integer, Integer, Integer, Integer, Integer,
Integer, Tuple(Of Integer, Integer, Integer))) _
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910)

        ShowComponentCount(population)
   End Sub

   Private Sub ShowComponentCount(tuple As Object)
      Static ctr As Integer = 0
      Static totalComponents As Integer = 0
      Dim components As Integer = 0

      ctr += 1
      Dim name As String = tuple.GetType().Name
      components += Int32.Parse(name.Substring(name.IndexOf("`") + 1))
      If components = 8 Then
         totalComponents += 7
         Console.WriteLine("The tuple at level {0} has 7 components.", ctr)
         ShowComponentCount(tuple.Rest)
      Else
         totalComponents += components
         Console.WriteLine("The tuple at level {0} has {1} components.",
                           ctr, components)
         Console.WriteLine("Total components in tuple: {0}",
totalComponents)
      End If
   End Sub
End Module
' The example displays the following output:
'       The tuple at level 1 has 7 components.
'       The tuple at level 2 has 7 components.
'       The tuple at level 3 has 3 components.
'       Total components in tuple: 17
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Examples

The following example defines five Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects that contain prime numbers. It then compares the first object with each of the remaining objects. As the output shows, only the first and the last Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects are equal, because they have an identical number of components with identical values.

C#

```
using System;

public class Class1
```

```
{
   public static void Main()
   {
      // Create five 8-tuple objects containing prime numbers.
      var prime1 = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                           Tuple<Int32>> (2, 3, 5, 7, 11, 13, 17,
                           new Tuple<Int32>(19));
      var prime2 = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                           Tuple<Int32>> (23, 29, 31, 37, 41, 43, 47,
                           new Tuple<Int32>(55));
      var prime3 = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                           Tuple<Int32>> (3, 2, 5, 7, 11, 13, 17,
                           new Tuple<Int32>(19));
      var prime4 = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                           Tuple<Int32, Int32>> (2, 3, 5, 7, 11, 13, 17,
                           new Tuple<Int32, Int32>(19, 23));
      var prime5 = new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                           Tuple<Int32>> (2, 3, 5, 7, 11, 13, 17,
                           new Tuple<Int32>(19));
      Console.WriteLine("{0} = {1} : {2}", prime1, prime2,
prime1.Equals(prime2));
      Console.WriteLine("{0} = {1} : {2}", prime1, prime3,
prime1.Equals(prime3));
      Console.WriteLine("{0} = {1} : {2}", prime1, prime4,
prime1.Equals(prime4));
      Console.WriteLine("{0} = {1} : {2}", prime1, prime5,
prime1.Equals(prime5));
   }
}
// The example displays the following output:
//    (2, 3, 5, 7, 11, 13, 17, 19) = (23, 29, 31, 37, 41, 43, 47, 55) :
False
//    (2, 3, 5, 7, 11, 13, 17, 19) = (3, 2, 5, 7, 11, 13, 17, 19) : False
//    (2, 3, 5, 7, 11, 13, 17, 19) = (2, 3, 5, 7, 11, 13, 17, 19, 23) :
False
//    (2, 3, 5, 7, 11, 13, 17, 19) = (2, 3, 5, 7, 11, 13, 17, 19) : True
```

Remarks

The `obj` parameter is considered to be equal to the current instance if it meets all the following conditions:

- It is a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.
- It has the same total number of components that are of the same types as the current instance.

- Its components (including its nested components) are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Examples

The following example creates a 17-tuple that contains population data for the city of Detroit, Michigan, from 1860 to 1900. It then uses the ToString method to display the tuple's data.

C#

```
using System;

class Example
{
    static void Main(string[] args)
    {
        Tuple<int, int, int> from1980 = Tuple.Create(1203339, 1027974,
951270);
        var from1910 = new Tuple<int, int, int, int, int, int, int,
Tuple<int, int, int>>
            (465766, 993078, 1568622, 1623452, 1849568, 1670144, 1511462,
from1980);
```

```
        var population = new Tuple<string, int, int, int, int, int, int,
            Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>>
            ("Detroit", 1860, 45619, 79577, 116340, 205876, 285704,
from1910);

        Console.WriteLine(population.ToString());
    }

    private static void ShowPopulationChange(int year, int newPopulation,
int oldPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation,
                          ((double)(newPopulation - oldPopulation) /
oldPopulation) / 10);
    }

    private static void ShowPopulation(int year, int newPopulation)
    {
        Console.WriteLine("{0,5}  {1,14:N0}  {2,10:P2}", year,
newPopulation, "n/a");
    }
}
// The example displays the following output:
//   (Detroit, 1860, 45619, 79577, 116340, 205876, 285704, 465766, 993078,
//    1568622, 1623452, 1849568, 1670144, 1511462, 1203339, 1027974, 951270)
```

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, `Item5, Item6, Item7`, *Item8*...), where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, and *Item7* represent the values of the Item1, Item2, Item3, Item4, Item5, Item6, and Item7 properties. *Item8* represents the value of the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object's `Next.Item1` property. The value of any additional nested components follow *Item8*. If any of the property values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

An object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Examples

The following example creates an array of Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects that contains population data for four U.S. cities from 1940 to 2000. The octuple's first component is the city name. The remaining six components represent the population at 10-year intervals from 1940 to 2000.

The `PopulationComparer` class provides an IComparer implementation that allows the array of octuples to be sorted by any one of its components. Two values are provided to the `PopulationComparer` class in its constructor: The position of the component that defines the sort order, and a Boolean value that indicates whether the tuple objects should be sorted in ascending or descending order.

The example then displays the elements in the array in unsorted order, sorts them by the third component (the population in 1950) and displays them, and then sorts them by the eighth component (the population in 2000) and displays them.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class PopulationComparer<T1, T2, T3, T4, T5, T6, T7, T8> : IComparer
{
   private int itemPosition;
   private int multiplier = -1;

   public PopulationComparer(int component) : this(component, true)
   { }

   public PopulationComparer(int component, bool descending)
```

```
   {
      if (! descending) multiplier = 1;

      if (component <= 0 || component > 8)
         throw new ArgumentException("The component argument is out of
range.");

      itemPosition = component;
   }

   public int Compare(object x, object y)
   {
      Tuple<T1, T2, T3, T4, T5, T6, T7, Tuple<T8>> tX = x as Tuple<T1, T2,
T3, T4, T5, T6, T7, Tuple<T8>>;
      if (tX == null)
         return 0;

      Tuple<T1, T2, T3, T4, T5, T6, T7, Tuple<T8>> tY = y as Tuple<T1, T2,
T3, T4, T5, T6, T7, Tuple<T8>>;
      switch (itemPosition)
      {
         case 1:
            return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
         case 2:
            return Comparer<T2>.Default.Compare(tX.Item2, tY.Item2) *
multiplier;
         case 3:
            return Comparer<T3>.Default.Compare(tX.Item3, tY.Item3) *
multiplier;
         case 4:
            return Comparer<T4>.Default.Compare(tX.Item4, tY.Item4) *
multiplier;
         case 5:
            return Comparer<T5>.Default.Compare(tX.Item5, tY.Item5) *
multiplier;
         case 6:
            return Comparer<T6>.Default.Compare(tX.Item6, tY.Item6) *
multiplier;
         case 7:
            return Comparer<T7>.Default.Compare(tX.Item7, tY.Item7) *
multiplier;
         case 8:
            return Comparer<T8>.Default.Compare(tX.Rest.Item1,
tY.Rest.Item1) * multiplier;
         default:
            return Comparer<T1>.Default.Compare(tX.Item1, tY.Item1) *
multiplier;
      }
   }
}

public class Example
{
   public static void Main()
```

```
   {
      // Create array of octuples with population data for three U.S.
      // cities, 1940-2000.
      Tuple<string, int, int, int, int, int, int, Tuple<int>>[] cities  =
          { Tuple.Create("Los Angeles", 1504277, 1970358, 2479015, 2816061,
2966850, 3485398, 3694820),
            Tuple.Create("New York", 7454995, 7891957, 7781984, 7894862,
7071639, 7322564, 8008278),
            Tuple.Create("Chicago", 3396808, 3620962, 3550904, 3366957,
3005072, 2783726, 2896016),
            Tuple.Create("Detroit", 1623452, 1849568, 1670144, 1511462,
1203339, 1027974, 951270) };
      // Display array in unsorted order.
      Console.WriteLine("In unsorted order:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int, int, int>(2));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 1950:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
      Console.WriteLine();

      Array.Sort(cities, new PopulationComparer<string, int, int, int, int,
int, int, int>(8));

      // Display array in sorted order.
      Console.WriteLine("Sorted by population in 2000:");
      foreach (var city in cities)
         Console.WriteLine(city.ToString());
   }
}
// The example displays the following output:
//    In unsorted order:
//    (Los Angeles, 1504277, 1970358, 2479015, 2816061, 2966850, 3485398,
3694820)
//    (New York, 7454995, 7891957, 7781984, 7894862, 7071639, 7322564,
8008278)
//    (Chicago, 3396808, 3620962, 3550904, 3366957, 3005072, 2783726,
2896016)
//    (Detroit, 1623452, 1849568, 1670144, 1511462, 1203339, 1027974,
951270)
//
//    Sorted by population in 1950:
//    (New York, 7454995, 7891957, 7781984, 7894862, 7071639, 7322564,
8008278)
//    (Chicago, 3396808, 3620962, 3550904, 3366957, 3005072, 2783726,
2896016)
//    (Detroit, 1623452, 1849568, 1670144, 1511462, 1203339, 1027974,
951270)
//    (Los Angeles, 1504277, 1970358, 2479015, 2816061, 2966850, 3485398,
```

```
3694820)
//
//    Sorted by population in 2000:
//    (New York, 7454995, 7891957, 7781984, 7894862, 7071639, 7322564,
8008278)
//    (Los Angeles, 1504277, 1970358, 2479015, 2816061, 2966850, 3485398,
3694820)
//    (Chicago, 3396808, 3620962, 3550904, 3366957, 3005072, 2783726,
2896016)
//    (Detroit, 1623452, 1849568, 1670144, 1511462, 1203339, 1027974,
951270)
```

Remarks

This member is an explicit interface implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralComparable interface.

This method lets you define customized comparisons of Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects. For example, you can use this method to order Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects based on the value of a specific component.

Although this method can be called directly, it is most commonly called by collection-sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralEquatable

interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast (in C#) or converted (in Visual Basic) to a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object that has the same total number of components (including those in nested `Tuple` objects) of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects. This continues until the method call returns `false` when it compares a specific pair of values, or the two Rest values are passed to the method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralEquatable interface.

The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object obj);
```

Parameters

obj Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `obj`.
Zero	This instance and `obj` have the same position in the sort order.
A positive integer	This instance follows `obj`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`obj` is not a Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Examples

The following example creates an array of octuples whose components are integers that contain a range of prime numbers. The example displays the elements of the array in unsorted order, sorts the array, and then displays the array in sorted order. The output shows that the array has been sorted by Item1, or the tuple's first component. Note that the example does not directly call the IComparable.CompareTo(Object) method. This method is called implicitly by the Sort(Array) method for each element in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      // Create array of 8-tuple objects containing prime numbers.
      Tuple<Int32, Int32, Int32, Int32, Int32, Int32, Int32, Tuple<Int32>>[]
primes =
                     { new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                         Tuple<Int32>>(2, 3, 5, 7, 11, 13, 17, new
Tuple<Int32>(19)),
                       new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                         Tuple<Int32>>(23, 29, 31, 37, 41, 43, 47, new
Tuple<Int32>(55)),
                       new Tuple<Int32, Int32, Int32, Int32, Int32, Int32,
Int32,
                         Tuple<Int32>>(3, 2, 5, 7, 11, 13, 17, new
Tuple<Int32>(19)) };
      // Display 8-tuples in unsorted order.
      foreach (var prime in primes)
         Console.WriteLine(prime.ToString());
      Console.WriteLine();

      // Sort the array and display its 8-tuples.
      Array.Sort(primes);
      foreach (var prime in primes)
         Console.WriteLine(prime.ToString());
   }
```

```
}
// The example displays the following output:
//       (2, 3, 5, 7, 11, 13, 17, 19)
//       (23, 29, 31, 37, 41, 43, 47, 55)
//       (3, 2, 5, 7, 11, 13, 17, 19)
//
//       (2, 3, 5, 7, 11, 13, 17, 19)
//       (3, 2, 5, 7, 11, 13, 17, 19)
//       (23, 29, 31, 37, 41, 43, 47, 55)
```

Remarks

This member is an explicit interface member implementation. It can be used only when the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> class. Although the method can be called directly, it is most commonly called by the default overloads of collection-sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> ⊗ **Caution**
>
> The **IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Equals(Object)** method.

This method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `Tuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `Tuple` element. `index` can range from 0 for `Item1` to one less than the number of elements in the `Tuple`.

Property Value

Object

The value of the `Tuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0.

-or-

`index` is greater than or equal to ITuple.Length.

Remarks

The Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `Tuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

The number of elements in the `Tuple`.

Implements

Length

Remarks

The ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the Tuple<T1,T2,T3,T4,T5,T6,T7,TRest> type to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TupleExtensions Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides extension methods for tuples to interoperate with language support for tuples in C#.

C#

```
public static class TupleExtensions
```

Inheritance Object → TupleExtensions

Remarks

Although these methods can be called directly, they are primarily intended for use by the C# compiler in creating and deconstructing tuples.

Methods

Expand table

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21)	Deconstructs a tuple with 21 elements into separate variables.

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10, T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20)	Deconstructs a tuple with 20 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19)	Deconstructs a tuple with 19 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18)	Deconstructs a tuple with 18 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17)	Deconstructs a tuple with 17 elements

	into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14, Tuple<T15,T16>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16)	Deconstructs a tuple with 16 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15)	Deconstructs a tuple with 15 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14)	Deconstructs a tuple with 14 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10,	Deconstru

T11, T12, T13)	cts a tuple with 13 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12)	Deconstructs a tuple with 12 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11)	Deconstructs a tuple with 11 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10)	Deconstructs a tuple with 10 elements into sepa

	rate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>, T1, T2, T3, T4, T5, T6, T7, T8, T9)	Deconstructs a tuple with 9 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>, T1, T2, T3, T4, T5, T6, T7, T8)	Deconstructs a tuple with 8 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>, T1, T2, T3, T4, T5, T6, T7)	Deconstructs a tuple with 7 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>, T1, T2, T3, T4, T5, T6)	Deconstructs a tuple

	with 6 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>, T1, T2, T3, T4, T5)	Deconstructs a tuple with 5 elements into separate variables.
Deconstruct<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>, T1, T2, T3, T4)	Deconstructs a tuple with 4 elements into separate variables.
Deconstruct<T1,T2,T3>(Tuple<T1,T2,T3>, T1, T2, T3)	Deconstructs a tuple with 3 elements into separate

	variables.
Deconstruct<T1,T2>(Tuple<T1,T2>, T1, T2)	Deconstructs a tuple with 2 elements into separate variables.
Deconstruct<T1>(Tuple<T1>, T1)	Deconstructs a tuple with 1 element into a separate variable.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20,T21>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8, T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tupl`

	`e` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8, T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9, T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10, T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>)	Converts an instance

	of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11, T12,T13,T14,ValueTuple<T15,T16,T17>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(ValueTuple<T1, T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15, T16>>>)	Converts an instance of the `ValueTuple` structure

	to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tupl`

	`e` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>)	Converts an instance

	of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>)	Converts an instance of the `ValueTuple` structure

	to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7>(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tupl`

	`e` class.
ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5>(ValueTuple<T1,T2,T3,T4,T5>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)	Converts an instance

	of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2>(ValueTuple<T1,T2>)	Converts an instance of the `ValueTuple` structure

	to an instance of the `Tuple` class.
ToTuple<T1>(ValueTuple<T1>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple`

	structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10, T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>)	Converts an instance

	of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>)	Converts an instance of the `Tuple` class to an insta

	nce of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple`

	structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>)	Converts an instance

	of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>)	Converts an instance of the `Tuple` class to an insta

	nce of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple`

	structure.
ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)	Converts an instance

	of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3>(Tuple<T1,T2,T3>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2>(Tuple<T1,T2>)	Converts an instance of the `Tuple` class to an insta

	nce of the `ValueTuple` structure.
ToValueTuple<T1>(Tuple<T1>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

Applies to

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

TupleExtensions.Deconstruct Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Deconstructs the components of a tuple into separate variables.

Overloads

Expand table

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8, T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21)	Deconstructs a tuple with 21 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10, T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20)	Deconstructs a tuple with 20 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19)	Deconstructs a tuple with 19 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18)	Deconstructs a tuple with 18 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17)	Deconstructs a tuple with 17 elements into separate variables.

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14, Tuple<T15,T16>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16)	Deconstructs a tuple with 16 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15)	Deconstructs a tuple with 15 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14)	Deconstructs a tuple with 14 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13)	Deconstructs a tuple with 13 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1, T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12)	Deconstructs a tuple with 12 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11)	Deconstructs a tuple with 11 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10)	Deconstructs a tuple with 10 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>, T1, T2, T3, T4, T5, T6, T7, T8, T9)	Deconstructs a tuple with 9 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>, T1, T2, T3, T4, T5, T6, T7, T8)	Deconstructs a tuple with 8 elements into

	separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>, T1, T2, T3, T4, T5, T6, T7)	Deconstructs a tuple with 7 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>, T1, T2, T3, T4, T5, T6)	Deconstructs a tuple with 6 elements into separate variables.
Deconstruct<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>, T1, T2, T3, T4, T5)	Deconstructs a tuple with 5 elements into separate variables.
Deconstruct<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>, T1, T2, T3, T4)	Deconstructs a tuple with 4 elements into separate variables.
Deconstruct<T1,T2,T3>(Tuple<T1,T2,T3>, T1, T2, T3)	Deconstructs a tuple with 3 elements into separate variables.
Deconstruct<T1,T2>(Tuple<T1,T2>, T1, T2)	Deconstructs a tuple with 2 elements into separate variables.
Deconstruct<T1>(Tuple<T1>, T1)	Deconstructs a tuple with 1 element into a separate variable.

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11, T12,T13,T14,T15, T16,T17,T18,T19,T20,T21>

(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21)

Deconstructs a tuple with 21 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>> value, out T1 item1, out T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out T10 item10, out T11 item11, out T12 item12, out T13 item13, out T14 item14, out T15 item15, out T16 item16, out T17 item17, out T18 item18, out T19 item19, out T20 item20, out T21 item21);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

`T17`

The type of the seventeenth element.

`T18`

The type of the eighteenth element.

`T19`

The type of the nineteenth element.

`T20`

The type of the twentieth element.

`T21`

The type of the twenty-first element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>

The 21-element tuple to deconstruct into 21 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

`item17` T17

When this method returns, contains the value of the seventeenth element, or `value.Rest.Rest.Item3`.

`item18` T18

When this method returns, contains the value of the eighteenth element, or `value.Rest.Rest.Item4`.

`item19` T19

When this method returns, contains the value of the nineteenth element, or `value.Rest.Rest.Item5`.

`item20` T20

When this method returns, contains the value of the twentieth element, or `value.Rest.Rest.Item6`.

`item21` T21

When this method returns, contains the value of the twenty-first element, or `value.Rest.Rest.Item7`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item21` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20)

Deconstructs a tuple with 20 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>> value, out T1 item1, out T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out T10 item10, out T11 item11, out T12 item12, out T13 item13, out T14 item14, out T15 item15, out T16 item16, out T17 item17, out T18 item18, out T19 item19, out T20 item20);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

`T17`

The type of the seventeenth element.

`T18`

The type of the eighteenth element.

`T19`

The type of the nineteenth element.

`T20`

The type of the twentieth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>

The 20-element tuple to deconstruct into 20 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

`item17` T17

When this method returns, contains the value of the seventeenth element, or `value.Rest.Rest.Item3`.

`item18` T18

When this method returns, contains the value of the eighteenth element, or `value.Rest.Rest.Item4`.

`item19` T19

When this method returns, contains the value of the nineteenth element, or `value.Rest.Rest.Item5`.

`item20` T20

When this method returns, contains the value of the twentieth element, or `value.Rest.Rest.Item6`.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19)

Deconstructs a tuple with 19 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
```

```
T17,T18,T19>>> value, out T1 item1, out T2 item2, out T3 item3, out T4
item4, out T5 item5, out T6 item6, out T7 item7, out T8 item8, out T9
item9, out T10 item10, out T11 item11, out T12 item12, out T13 item13,
out T14 item14, out T15 item15, out T16 item16, out T17 item17, out T18
item18, out T19 item19);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

T9

The type of the ninth element.

T10

The type of the tenth element.

T11

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

`T17`

The type of the seventeenth element.

`T18`

The type of the eighteenth element.

`T19`

The type of the nineteenth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>

The 19-element tuple to deconstruct into 19 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

`item17` T17

When this method returns, contains the value of the seventeenth element, or `value.Rest.Rest.Item3`.

`item18` T18

When this method returns, contains the value of the eighteenth element, or `value.Rest.Rest.Item4`.

`item19` T19

When this method returns, contains the value of the nineteenth element, or `value.Rest.Rest.Item5`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item19` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11, T12,T13,T14,T15, T16,T17,T18> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18)

Deconstructs a tuple with 18 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T1
8> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17,T18>>> value, out T1 item1, out T2 item2, out T3 item3, out T4 item4,
out T5 item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out
T10 item10, out T11 item11, out T12 item12, out T13 item13, out T14
item14, out T15 item15, out T16 item16, out T17 item17, out T18 item18);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

`T17`

The type of the seventeenth element.

`T18`

The type of the eighteenth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18>>>

The 18-element tuple to deconstruct into 18 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

`item17` T17

When this method returns, contains the value of the seventeenth element, or `value.Rest.Rest.Item3`.

`item18` T18

When this method returns, contains the value of the eighteenth element, or `value.Rest.Rest.Item4`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item18` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10, T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11, T12,T13,T14,T15, T16,T17>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17)

Deconstructs a tuple with 17 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17>
(this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
```

```
T17>>> value, out T1 item1, out T2 item2, out T3 item3, out T4 item4, out
T5 item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out T10
item10, out T11 item11, out T12 item12, out T13 item13, out T14 item14,
out T15 item15, out T16 item16, out T17 item17);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

T9

The type of the ninth element.

T10

The type of the tenth element.

T11

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

`T17`

The type of the seventeenth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>>

The 17-element tuple to deconstruct into 17 separate variables.

`item1`	T1

When this method returns, contains the value of the first element.

`item2`	T2

When this method returns, contains the value of the second element.

`item3`	T3

When this method returns, contains the value of the third element.

`item4`	T4

When this method returns, contains the value of the fourth element.

`item5`	T5

When this method returns, contains the value of the fifth element.

`item6`	T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

`item17` T17

When this method returns, contains the value of the seventeenth element, or `value.Rest.Rest.Item3`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item17` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T1

1,T12,T13,T14, Tuple<T15,T16>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16)

Deconstructs a tuple with 16 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>
>> value, out T1 item1, out T2 item2, out T3 item3, out T4 item4, out T5
item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out T10
item10, out T11 item11, out T12 item12, out T13 item13, out T14 item14,
out T15 item15, out T16 item16);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

`T16`

The type of the sixteenth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>

The 16-element tuple to deconstruct into 16 separate variables.

`item1`	T1

When this method returns, contains the value of the first element.

`item2`	T2

When this method returns, contains the value of the second element.

`item3`	T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

`item16` T16

When this method returns, contains the value of the sixteenth element, or `value.Rest.Rest.Item2`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item16` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1,

T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15)

Deconstructs a tuple with 15 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>
value, out T1 item1, out T2 item2, out T3 item3, out T4 item4, out T5
item5, out T6 item6, out T7 item7, out T8 item8, out T9 item9, out T10
item10, out T11 item11, out T12 item12, out T13 item13, out T14 item14,
out T15 item15);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

`T15`

The type of the fifteenth element.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>

The 15-element tuple to deconstruct into 15 separate variables.

`item1`	T1

When this method returns, contains the value of the first element.

`item2`	T2

When this method returns, contains the value of the second element.

`item3`	T3

When this method returns, contains the value of the third element.

`item4`	T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

`item15` T15

When this method returns, contains the value of the fifteenth element, or `value.Rest.Rest.Item1`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item15` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14)

Deconstructs a tuple with 14 elements into separate variables.

```
C#
```

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>> value, out
T1 item1, out T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6
item6, out T7 item7, out T8 item8, out T9 item9, out T10 item10, out T11
item11, out T12 item12, out T13 item13, out T14 item14);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

T9

The type of the ninth element.

T10

The type of the tenth element.

T11

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

`T14`

The type of the fourteenth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>

The 14-element tuple to deconstruct into 14 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

`item14` T14

When this method returns, contains the value of the fourteenth element, or `value.Rest.Item7`.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13)

Deconstructs a tuple with 13 elements into separate variables.

C#

```
public static void
Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>> value, out T1
item1, out T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6
item6, out T7 item7, out T8 item8, out T9 item9, out T10 item10, out T11
item11, out T12 item12, out T13 item13);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

`T13`

The type of the thirteenth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>

The 13-element tuple to deconstruct into 13 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

`item13` T13

When this method returns, contains the value of the thirteenth element, or `value.Rest.Item6`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8`

through `item13` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12)

Deconstructs a tuple with 12 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>
(this Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>> value, out T1
item1, out T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6
item6, out T7 item7, out T8 item8, out T9 item9, out T10 item10, out T11
item11, out T12 item12);
```

Type Parameters

T1

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

`T12`

The type of the twelfth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>

The 12-element tuple to deconstruct into 12 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

`item12` T12

When this method returns, contains the value of the twelfth element, or `value.Rest.Item5`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item12` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11)

Deconstructs a tuple with 11 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>> value, out T1 item1, out
T2 item2, out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7
item7, out T8 item8, out T9 item9, out T10 item10, out T11 item11);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

`T11`

The type of the eleventh element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>

The 11-element tuple to deconstruct into 11 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

`item11` T11

When this method returns, contains the value of the eleventh element, or `value.Rest.Item4`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item11` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10)

Deconstructs a tuple with 10 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>> value, out T1 item1, out T2
item2, out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7
item7, out T8 item8, out T9 item9, out T10 item10);
```

Type Parameters

T1

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

`T10`

The type of the tenth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>

The 10-element tuple to deconstruct into 10 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

`item10` T10

When this method returns, contains the value of the tenth element, or `value.Rest.Item3`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` through `item10` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>)

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>, T1, T2, T3, T4, T5, T6, T7, T8, T9)

Deconstructs a tuple with 9 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8,T9> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>> value, out T1 item1, out T2
item2, out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7
item7, out T8 item8, out T9 item9);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element.

`T9`

The type of the ninth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>

The 9-element tuple to deconstruct into 9 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

`item9` T9

When this method returns, contains the value of the ninth element, or `value.Rest.Item2`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` and `item9` are elements of nested tuples.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>, T1, T2, T3, T4, T5, T6, T7, T8)

Deconstructs a tuple with 8 elements into separate variables.

```
C#

public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7,T8> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>> value, out T1 item1, out T2 item2,
out T3 item3, out T4 item4, out T5 item5, out T6 item6, out T7 item7, out
T8 item8);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element.

Parameters

value Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>

The 8-element tuple to deconstruct into 8 separate variables.

item1 T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

`item8` T8

When this method returns, contains the value of the eighth element, or `value.Rest.Item1`.

Remarks

This method is implemented primarily to support the tuple language features in C#.

Because the .NET Framework tuple types implement tuples with more than 7 elements by nesting a tuple in Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest, `item8` is an element in a nested tuple.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6,T7> (Tuple<T1,T2,T3,T4,T5,T6,T7>, T1, T2, T3, T4, T5, T6, T7)

Deconstructs a tuple with 7 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6,T7> (this
Tuple<T1,T2,T3,T4,T5,T6,T7> value, out T1 item1, out T2 item2, out T3
item3, out T4 item4, out T5 item5, out T6 item6, out T7 item7);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7>

The 7-element tuple to deconstruct into 7 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

`item7` T7

When this method returns, contains the value of the seventh element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7>(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)
- ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5,T6> (Tuple<T1,T2,T3,T4,T5,T6>, T1, T2, T3, T4, T5, T6)

Deconstructs a tuple with 6 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5,T6> (this
Tuple<T1,T2,T3,T4,T5,T6> value, out T1 item1, out T2 item2, out T3 item3,
out T4 item4, out T5 item5, out T6 item6);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6>

The 6-element tuple to deconstruct into 6 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

`item6` T6

When this method returns, contains the value of the sixth element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)
- ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4,T5> (Tuple<T1,T2,T3,T4,T5>, T1, T2, T3, T4, T5)

Deconstructs a tuple with 5 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3,T4,T5> (this
Tuple<T1,T2,T3,T4,T5> value, out T1 item1, out T2 item2, out T3 item3,
out T4 item4, out T5 item5);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

Parameters

value Tuple<T1,T2,T3,T4,T5>

The 5-element tuple to deconstruct into 5 separate variables.

item1 T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

`item5` T5

When this method returns, contains the value of the fifth element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2,T3,T4,T5>(ValueTuple<T1,T2,T3,T4,T5>)
- ToValueTuple<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>, T1, T2, T3, T4)

Deconstructs a tuple with 4 elements into separate variables.

```
C#
```

```
public static void Deconstruct<T1,T2,T3,T4> (this Tuple<T1,T2,T3,T4>
value, out T1 item1, out T2 item2, out T3 item3, out T4 item4);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

Parameters

`value` Tuple<T1,T2,T3,T4>

The 4-element tuple to deconstruct into 4 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

`item4` T4

When this method returns, contains the value of the fourth element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)
- ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2,T3>(Tuple<T1,T2,T3>, T1, T2, T3)

Deconstructs a tuple with 3 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2,T3> (this Tuple<T1,T2,T3> value, out
T1 item1, out T2 item2, out T3 item3);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

Parameters

value Tuple<T1,T2,T3>

The 3-element tuple to deconstruct into 3 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

`item3` T3

When this method returns, contains the value of the third element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)
- ToValueTuple<T1,T2,T3>(Tuple<T1,T2,T3>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1,T2>(Tuple<T1,T2>, T1, T2)

Deconstructs a tuple with 2 elements into separate variables.

C#

```
public static void Deconstruct<T1,T2> (this Tuple<T1,T2> value, out T1
item1, out T2 item2);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

Parameters

`value` Tuple<T1,T2>

The 2-element tuple to deconstruct into 2 separate variables.

`item1` T1

When this method returns, contains the value of the first element.

`item2` T2

When this method returns, contains the value of the second element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1,T2>(ValueTuple<T1,T2>)
- ToValueTuple<T1,T2>(Tuple<T1,T2>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Deconstruct<T1>(Tuple<T1>, T1)

Deconstructs a tuple with 1 element into a separate variable.

```
C#
public static void Deconstruct<T1> (this Tuple<T1> value, out T1 item1);
```

Type Parameters

`T1`

The type of the single element.

Parameters

`value` Tuple<T1>

The 1-element tuple to deconstruct into a separate variable.

`item1` T1

When this method returns, contains the value of the single element.

Remarks

This method is implemented primarily to support the tuple language features in C#.

See also

- ToTuple<T1>(ValueTuple<T1>)
- ToValueTuple<T1>(Tuple<T1>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TupleExtensions.ToTuple Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a value tuple to a tuple.

Overloads

Expand table

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20,T21>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10, T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11, T12,T13,T14,ValueTuple<T15,T16,T17>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(ValueTuple<T1, T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15, T16>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6,T7>(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4,T5>(ValueTuple<T1,T2,T3,T4,T5>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1,T2>(ValueTuple<T1,T2>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.
ToTuple<T1>(ValueTuple<T1>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20,T21>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>
```

```
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19
,T20,T21> (this (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17, T18, T19, T20, T21))) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

`T19`

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

`T20`

The type of the twentieth element, or `value.Rest.Rest.Item6`.

`T21`

The type of the twenty-first element, or `value.Rest.Rest.Item7`.

Parameters

`value`	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19,T20>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20> (this (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15, T16, T17, T18, T19, T20))) value);
```

Type Parameters

T1

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

`T19`

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

`T20`

The type of the twentieth element, or `value.Rest.Rest.Item6`.

Parameters

`value`	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9, T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19> (this (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15, T16, T17, T18, T19))) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

`T19`

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

Parameters

value	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8

,T9,T10, T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T
17,T18>>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18>
(this (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15,
T16, T17, T18))) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

Parameters

`value`	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11, T12,T13,T14,ValueTuple<T15,T16,T17>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>> ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17>(this (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15, T16, T17))) value);
```

Type Parameters

T1

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

T16

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

T17

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

Parameters

value	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(ValueTuple<T1,

T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15, T16>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>
> ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> (this
(T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15,
T16))) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

T10

The type of the tenth element, or `value.Rest.Item3`.

T11

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

T16

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

Parameters

value	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> (this (T1, T2,
T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14, (T15))) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

T10

The type of the tenth element, or `value.Rest.Item3`.

T11

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

Parameters

value	ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (this (T1, T2, T3,
T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13, T14)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> (this (T1, T2, T3, T4,
T5, T6, T7, (T8, T9, T10, T11, T12, T13)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> (this (T1, T2, T3, T4, T5,
T6, T7, (T8, T9, T10, T11, T12)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> (this (T1, T2, T3, T4, T5, T6,
T7, (T8, T9, T10, T11)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> (this (T1, T2, T3, T4, T5, T6, T7,
(T8, T9, T10)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9> (this (T1, T2, T3, T4, T5, T6, T7,
(T8, T9)) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7,T8>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>
ToTuple<T1,T2,T3,T4,T5,T6,T7,T8> (this (T1, T2, T3, T4, T5, T6, T7, (T8))
value);
```

Type Parameters

`T1`

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

Parameters

value ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6,T7> (ValueTuple<T1,T2,T3,T4,T5,T6,T7>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3,T4,T5,T6,T7> ToTuple<T1,T2,T3,T4,T5,T6,T7>
(this (T1, T2, T3, T4, T5, T6, T7) value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6,T7>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6,T7>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

C#

```
public static Tuple<T1,T2,T3,T4,T5,T6> ToTuple<T1,T2,T3,T4,T5,T6> (this
(T1, T2, T3, T4, T5, T6) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5,T6>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5,T6>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4,T5> (ValueTuple<T1,T2,T3,T4,T5>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3,T4,T5> ToTuple<T1,T2,T3,T4,T5> (this (T1, T2,
T3, T4, T5) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

Parameters

`value` ValueTuple<T1,T2,T3,T4,T5>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4,T5>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3,T4> ToTuple<T1,T2,T3,T4> (this (T1, T2, T3,
T4) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

Parameters

`value` ValueTuple<T1,T2,T3,T4>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3,T4>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2,T3> ToTuple<T1,T2,T3> (this (T1, T2, T3) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

Parameters

`value` ValueTuple<T1,T2,T3>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2,T3>

The converted tuple.

See also

- ToValueTuple<T1,T2,T3>(Tuple<T1,T2,T3>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1,T2>(ValueTuple<T1,T2>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1,T2> ToTuple<T1,T2> (this (T1, T2) value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

Parameters

`value` ValueTuple<T1,T2>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1,T2>

The converted tuple.

See also

- ToValueTuple<T1,T2>(Tuple<T1,T2>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToTuple<T1>(ValueTuple<T1>)

Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

```
C#

public static Tuple<T1> ToTuple<T1> (this (T1) value);
```

Type Parameters

`T1`

The type of the first element.

Parameters

`value` ValueTuple<T1>

The value tuple instance to convert to a tuple.

Returns

Tuple<T1>

The converted tuple.

See also

- ToValueTuple<T1>(Tuple<T1>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TupleExtensions.ToValueTuple Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a tuple to a value tuple.

Overloads

Expand table

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8, T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10, T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>)	Converts an instance of the `Tuple`

	class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `Tuple` class to an instance of

	the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4,T5>(Tuple<T1,T2,T3,T4,T5>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2,T3>(Tuple<T1,T2,T3>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.
ToValueTuple<T1,T2>(Tuple<T1,T2>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

ToValueTuple<T1>(Tuple<T1>)	Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20,T21>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17, T18, T19, T20, T21)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T
18,T19,T20,T21> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17,T18,T19,T20,T21>>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

`T19`

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

`T20`

The type of the twentieth element, or `value.Rest.Rest.Item6`.

`T21`

The type of the twenty-first element, or `value.Rest.Rest.Item7`.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20,T21>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20,T21>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19,T20>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17, T18, T19, T20)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T
18,T19,T20> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17,T18,T19,T20>>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

`T19`

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

`T20`

The type of the twentieth element, or `value.Rest.Rest.Item6`.

Parameters

value	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19,T20>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19,T20>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18,T19>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11, T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17, T18, T19)))
```

```
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T
18,T19> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17,T18,T19>>> value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

T16

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

T17

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

T18

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

T19

The type of the nineteenth element, or `value.Rest.Rest.Item5`.

Parameters

value	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18,T19>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18,T19>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9, T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18,T19>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17,T18>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12, T13,T14,Tuple<T15,T16,T17,T18>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17, T18)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T
18> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17,T18>>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

`T17`

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

`T18`

The type of the eighteenth element, or `value.Rest.Rest.Item4`.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17,T18>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17,T18>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17,T18>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15, T16,T17>

(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13, T14,Tuple<T15,T16,T17>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16, T17)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17>
(this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,
T17>>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

T10

The type of the tenth element, or `value.Rest.Item3`.

T11

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

T16

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

T17

The type of the seventeenth element, or `value.Rest.Rest.Item3`.

Parameters

value	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16,T17>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16, T17>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16,T17>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15, T16)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>
(this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>
>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

`T14`

The type of the fourteenth element, or `value.Rest.Item7`.

`T15`

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

`T16`

The type of the sixteenth element, or `value.Rest.Rest.Item2`.

Parameters

`value`	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15,T16>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15,T16>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```C#
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14, (T15)))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>
value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

T11

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

T15

The type of the fifteenth element, or `value.Rest.Rest.Item1`.

Parameters

value	Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14,Tuple<T15>>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14,ValueTuple<T15>>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13,
T14)) ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

T10

The type of the tenth element, or `value.Rest.Item3`.

T11

The type of the eleventh element, or `value.Rest.Item4`.

T12

The type of the twelfth element, or `value.Rest.Item5`.

T13

The type of the thirteenth element, or `value.Rest.Item6`.

T14

The type of the fourteenth element, or `value.Rest.Item7`.

Parameters

value Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13,T14>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13,T14>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12, T13))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

`T13`

The type of the thirteenth element, or `value.Rest.Item6`.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12,T13>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12,T13>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11, T12))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

`T12`

The type of the twelfth element, or `value.Rest.Item5`.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11,T12>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11,T12>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10, T11))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

`T11`

The type of the eleventh element, or `value.Rest.Item4`.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10,T11>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10,T11>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9, T10))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

`T8`

The type of the eighth element, or `value.Rest.Item1`.

`T9`

The type of the ninth element, or `value.Rest.Item2`.

`T10`

The type of the tenth element, or `value.Rest.Item3`.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9,T10>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9,T10>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8, T9))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

T9

The type of the ninth element, or `value.Rest.Item2`.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8,T9>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8,T9>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8,T9>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8>(Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7, (T8))
ToValueTuple<T1,T2,T3,T4,T5,T6,T7,T8> (this
Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>> value);
```

Type Parameters

`T1`

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

T7

The type of the seventh element.

T8

The type of the eighth element, or `value.Rest.Item1`.

Parameters

value Tuple<T1,T2,T3,T4,T5,T6,T7,Tuple<T8>>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7,T8>(ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6,T7>(Tuple<T1,T2,T3,T4,T5,T6,T7>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6, T7)
ToValueTuple<T1,T2,T3,T4,T5,T6,T7> (this Tuple<T1,T2,T3,T4,T5,T6,T7>
value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

T4

The type of the fourth element.

T5

The type of the fifth element.

T6

The type of the sixth element.

`T7`

The type of the seventh element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6,T7>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6,T7>(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5,T6>(Tuple<T1,T2,T3,T4,T5,T6>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5, T6) ToValueTuple<T1,T2,T3,T4,T5,T6>
(this Tuple<T1,T2,T3,T4,T5,T6> value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

`T6`

The type of the sixth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5,T6>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4,T5> (Tuple<T1,T2,T3,T4,T5>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4, T5) ToValueTuple<T1,T2,T3,T4,T5> (this
Tuple<T1,T2,T3,T4,T5> value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

`T4`

The type of the fourth element.

`T5`

The type of the fifth element.

Parameters

`value` Tuple<T1,T2,T3,T4,T5>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4,T5>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4,T5>(ValueTuple<T1,T2,T3,T4,T5>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3,T4>(Tuple<T1,T2,T3,T4>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3, T4) ToValueTuple<T1,T2,T3,T4> (this
Tuple<T1,T2,T3,T4> value);
```

Type Parameters

T1

The type of the first element.

T2

The type of the second element.

T3

The type of the third element.

`T4`

The type of the fourth element.

Parameters

`value` Tuple<T1,T2,T3,T4>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3,T4>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2,T3>(Tuple<T1,T2,T3>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2, T3) ToValueTuple<T1,T2,T3> (this Tuple<T1,T2,T3>
value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

`T3`

The type of the third element.

Parameters

`value` Tuple<T1,T2,T3>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2,T3>

The converted value tuple instance.

See also

- ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1,T2>(Tuple<T1,T2>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1, T2) ToValueTuple<T1,T2> (this Tuple<T1,T2> value);
```

Type Parameters

`T1`

The type of the first element.

`T2`

The type of the second element.

Parameters

`value` Tuple<T1,T2>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1,T2>

The converted value tuple instance.

See also

- ToTuple<T1,T2>(ValueTuple<T1,T2>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

ToValueTuple<T1>(Tuple<T1>)

Converts an instance of the `Tuple` class to an instance of the `ValueTuple` structure.

C#

```
public static (T1) ToValueTuple<T1> (this Tuple<T1> value);
```

Type Parameters

`T1`

The type of the first element.

Parameters

`value` Tuple<T1>

The tuple object to convert to a value tuple.

Returns

ValueTuple<T1>

The converted value tuple instance.

See also

- ToTuple<T1>(ValueTuple<T1>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types.

C#

```
public abstract class Type : System.Reflection.MemberInfo,
System.Reflection.IReflect
```

Inheritance Object → MemberInfo → Type

Derived System.Reflection.TypeInfo

Implements IReflect

Examples

The following example shows a few representative features of Type. The C# `typeof` operator (`GetType` operator in Visual Basic) is used to get a Type object representing String. From this Type object, the GetMethod method is used to get a MethodInfo representing the String.Substring overload that takes a starting location and a length.

To identify the overload signature, the code example creates a temporary array containing two Type objects representing `int` (`Integer` in Visual Basic).

The code example uses the MethodInfo to invoke the Substring method on the string "Hello, World!", and displays the result.

C#

```
using System;
using System.Reflection;
```

```
class Example
{
    static void Main()
    {
        Type t = typeof(String);

        MethodInfo substr = t.GetMethod("Substring",
            new Type[] { typeof(int), typeof(int) });

        Object result =
            substr.Invoke("Hello, World!", new Object[] { 7, 5 });
        Console.WriteLine("{0} returned \"{1}\".", substr, result);
    }
}

/* This code example produces the following output:

System.String Substring(Int32, Int32) returned "World".
 */
```

Remarks

For more information about this API, see Supplemental API remarks for Type.

Notes to Implementers

When you inherit from `Type`, you must override the following members:

- Assembly
- AssemblyQualifiedName
- BaseType
- FullName
- GetAttributeFlagsImpl()
- GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors(BindingFlags)
- GetElementType()
- GetEvent(String, BindingFlags)
- GetEvents(BindingFlags)
- GetField(String, BindingFlags)
- GetFields(BindingFlags)
- GetInterface(String, Boolean)
- GetInterfaces()

- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods(BindingFlags)
- GetNestedType(String, BindingFlags)
- GetNestedTypes(BindingFlags)
- GetProperties(BindingFlags)
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GUID
- HasElementTypeImpl()
- InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])
- IsArrayImpl()
- IsByRefImpl()
- IsCOMObjectImpl()
- IsPointerImpl()
- IsPrimitiveImpl()
- Module
- Namespace
- TypeHandle
- UnderlyingSystemType
- GetCustomAttributes(Boolean)
- GetCustomAttributes(Type, Boolean)
- IsDefined(Type, Boolean)
- Name

Constructors

Expand table

Type()	Initializes a new instance of the Type class.

Fields

Expand table

Delimiter	Separates names in the namespace of the Type. This field is read-only.
EmptyTypes	Represents an empty array of type Type. This field is read-only.
FilterAttribute	Represents the member filter used on attributes. This field is read-only.

FilterName	Represents the case-sensitive member filter used on names. This field is read-only.
FilterNameIgnoreCase	Represents the case-insensitive member filter used on names. This field is read-only.
Missing	Represents a missing value in the Type information. This field is read-only.

Properties

Expand table

Assembly	Gets the Assembly in which the type is declared. For generic types, gets the Assembly in which the generic type is defined.
AssemblyQualifiedName	Gets the assembly-qualified name of the type, which includes the name of the assembly from which this Type object was loaded.
Attributes	Gets the attributes associated with the Type.
BaseType	Gets the type from which the current Type directly inherits.
ContainsGenericParameters	Gets a value indicating whether the current Type object has type parameters that have not been replaced by specific types.
CustomAttributes	Gets a collection that contains this member's custom attributes. (Inherited from MemberInfo)
DeclaringMethod	Gets a MethodBase that represents the declaring method, if the current Type represents a type parameter of a generic method.
DeclaringType	Gets the type that declares the current nested type or generic type parameter.
DefaultBinder	Gets a reference to the default binder, which implements internal rules for selecting the appropriate members to be called by InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[]).
FullName	Gets the fully qualified name of the type, including its namespace but not its assembly.
GenericParameterAttributes	Gets a combination of GenericParameterAttributes flags that describe the covariance and special constraints of the current generic type parameter.
GenericParameterPosition	Gets the position of the type parameter in the type parameter list of the generic type or method that declared the parameter, when the Type object represents a type parameter of a generic type or a generic method.

GenericType Arguments	Gets an array of the generic type arguments for this type.
GUID	Gets the GUID associated with the Type.
HasElementType	Gets a value indicating whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
IsAbstract	Gets a value indicating whether the Type is abstract and must be overridden.
IsAnsiClass	Gets a value indicating whether the string format attribute `AnsiClass` is selected for the Type.
IsArray	Gets a value that indicates whether the type is an array.
IsAutoClass	Gets a value indicating whether the string format attribute `AutoClass` is selected for the Type.
IsAutoLayout	Gets a value indicating whether the fields of the current type are laid out automatically by the common language runtime.
IsByRef	Gets a value indicating whether the Type is passed by reference.
IsByRefLike	Gets a value that indicates whether the type is a byref-like structure.
IsClass	Gets a value indicating whether the Type is a class or a delegate; that is, not a value type or interface.
IsCollectible	Gets a value that indicates whether this MemberInfo object is part of an assembly held in a collectible AssemblyLoadContext. (Inherited from MemberInfo)
IsCOMObject	Gets a value indicating whether the Type is a COM object.
IsConstructed GenericType	Gets a value that indicates whether this object represents a constructed generic type. You can create instances of a constructed generic type.
IsContextful	Gets a value indicating whether the Type can be hosted in a context.
IsEnum	Gets a value indicating whether the current Type represents an enumeration.
IsExplicitLayout	Gets a value indicating whether the fields of the current type are laid out at explicitly specified offsets.
IsFunctionPointer	Gets a value that indicates whether the current Type is a function pointer.
IsGenericMethod Parameter	Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic method.
IsGeneric Parameter	Gets a value indicating whether the current Type represents a type parameter in the definition of a generic type or method.

IsGenericType	Gets a value indicating whether the current type is a generic type.
IsGenericType Definition	Gets a value indicating whether the current Type represents a generic type definition, from which other generic types can be constructed.
IsGenericType Parameter	Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic type.
IsImport	Gets a value indicating whether the Type has a ComImportAttribute attribute applied, indicating that it was imported from a COM type library.
IsInterface	Gets a value indicating whether the Type is an interface; that is, not a class or a value type.
IsLayout Sequential	Gets a value indicating whether the fields of the current type are laid out sequentially, in the order that they were defined or emitted to the metadata.
IsMarshalByRef	Gets a value indicating whether the Type is marshaled by reference.
IsNested	Gets a value indicating whether the current Type object represents a type whose definition is nested inside the definition of another type.
IsNested Assembly	Gets a value indicating whether the Type is nested and visible only within its own assembly.
IsNestedFam ANDAssem	Gets a value indicating whether the Type is nested and visible only to classes that belong to both its own family and its own assembly.
IsNestedFamily	Gets a value indicating whether the Type is nested and visible only within its own family.
IsNestedFam ORAssem	Gets a value indicating whether the Type is nested and visible only to classes that belong to either its own family or to its own assembly.
IsNestedPrivate	Gets a value indicating whether the Type is nested and declared private.
IsNestedPublic	Gets a value indicating whether a class is nested and declared public.
IsNotPublic	Gets a value indicating whether the Type is not declared public.
IsPointer	Gets a value indicating whether the Type is a pointer.
IsPrimitive	Gets a value indicating whether the Type is one of the primitive types.
IsPublic	Gets a value indicating whether the Type is declared public.
IsSealed	Gets a value indicating whether the Type is declared sealed.
IsSecurityCritical	Gets a value that indicates whether the current type is security-critical or security-safe-critical at the current trust level, and therefore can perform critical operations.

IsSecuritySafe Critical	Gets a value that indicates whether the current type is security-safe-critical at the current trust level; that is, whether it can perform critical operations and can be accessed by transparent code.
IsSecurity Transparent	Gets a value that indicates whether the current type is transparent at the current trust level, and therefore cannot perform critical operations.
IsSerializable	**Obsolete.** Gets a value indicating whether the Type is binary serializable.
IsSignatureType	Gets a value that indicates whether the type is a signature type.
IsSpecialName	Gets a value indicating whether the type has a name that requires special handling.
IsSZArray	Gets a value that indicates whether the type is an array type that can represent only a single-dimensional array with a zero lower bound.
IsTypeDefinition	Gets a value that indicates whether the type is a type definition.
IsUnicodeClass	Gets a value indicating whether the string format attribute `UnicodeClass` is selected for the Type.
IsUnmanaged FunctionPointer	Gets a value that indicates whether the current Type is an unmanaged function pointer.
IsValueType	Gets a value indicating whether the Type is a value type.
IsVariableBound Array	Gets a value that indicates whether the type is an array type that can represent a multi-dimensional array or an array with an arbitrary lower bound.
IsVisible	Gets a value indicating whether the Type can be accessed by code outside the assembly.
MemberType	Gets a MemberTypes value indicating that this member is a type or a nested type.
MetadataToken	Gets a value that identifies a metadata element. (Inherited from MemberInfo)
Module	Gets the module (the DLL) in which the current Type is defined.
Name	Gets the name of the current member. (Inherited from MemberInfo)
Namespace	Gets the namespace of the Type.
ReflectedType	Gets the class object that was used to obtain this member.
StructLayout	Gets a StructLayoutAttribute that describes the layout of the current type.

Attribute	
TypeHandle	Gets the handle for the current Type.
TypeInitializer	Gets the initializer for the type.
Underlying SystemType	Indicates the type provided by the common language runtime that represents this type.

Methods

Expand table

Equals(Object)	Determines if the underlying system type of the current Type object is the same as the underlying system type of the specified Object.
Equals(Type)	Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Type.
FindInterfaces(TypeFilter, Object)	Returns an array of Type objects representing a filtered list of interfaces implemented or inherited by the current Type.
FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)	Returns a filtered array of MemberInfo objects of the specified member type.
GetArrayRank()	Gets the number of dimensions in an array.
GetAttributeFlagsImpl()	When overridden in a derived class, implements the Attributes property and gets a bitwise combination of enumeration values that indicate the attributes associated with the Type.
GetConstructor(BindingFlags, Binder, Calling Conventions, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetConstructor(BindingFlags, Binder, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints.

GetConstructor(BindingFlags, Type[])	Searches for a constructor whose parameters match the specified argument types, using the specified binding constraints.
GetConstructor(Type[])	Searches for a public instance constructor whose parameters match the types in the specified array.
GetConstructorImpl(BindingFlags, Binder, Calling Conventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetConstructors()	Returns all the public constructors defined for the current Type.
GetConstructors(BindingFlags)	When overridden in a derived class, searches for the constructors defined for the current Type, using the specified `BindingFlags`.
GetCustomAttributes(Boolean)	When overridden in a derived class, returns an array of all custom attributes applied to this member. (Inherited from MemberInfo)
GetCustomAttributes(Type, Boolean)	When overridden in a derived class, returns an array of custom attributes applied to this member and identified by Type. (Inherited from MemberInfo)
GetCustomAttributesData()	Returns a list of CustomAttributeData objects representing data about the attributes that have been applied to the target member. (Inherited from MemberInfo)
GetDefaultMembers()	Searches for the members defined for the current Type whose DefaultMemberAttribute is set.
GetElementType()	When overridden in a derived class, returns the Type of the object encompassed or referred to by the current array, pointer or reference type.
GetEnumName(Object)	Returns the name of the constant that has the specified value, for the current enumeration type.

GetEnumNames()	Returns the names of the members of the current enumeration type.
GetEnumUnderlyingType()	Returns the underlying type of the current enumeration type.
GetEnumValues()	Returns an array of the values of the constants in the current enumeration type.
GetEnumValuesAsUnderlyingType()	Retrieves an array of the values of the underlying type constants of this enumeration type.
GetEvent(String)	Returns the EventInfo object representing the specified public event.
GetEvent(String, BindingFlags)	When overridden in a derived class, returns the EventInfo object representing the specified event, using the specified binding constraints.
GetEvents()	Returns all the public events that are declared or inherited by the current Type.
GetEvents(BindingFlags)	When overridden in a derived class, searches for events that are declared or inherited by the current Type, using the specified binding constraints.
GetField(String)	Searches for the public field with the specified name.
GetField(String, BindingFlags)	Searches for the specified field, using the specified binding constraints.
GetFields()	Returns all the public fields of the current Type.
GetFields(BindingFlags)	When overridden in a derived class, searches for the fields defined for the current Type, using the specified binding constraints.
GetFunctionPointerCallingConventions()	When overridden in a derived class, returns the calling conventions of the current function pointer Type.
GetFunctionPointerParameterTypes()	When overridden in a derived class, returns the parameter types of the current function pointer Type.
GetFunctionPointerReturnType()	When overridden in a derived class, returns the return type of the current function

	pointer Type.
GetGenericArguments()	Returns an array of Type objects that represent the type arguments of a closed generic type or the type parameters of a generic type definition.
GetGenericParameterConstraints()	Returns an array of Type objects that represent the constraints on the current generic type parameter.
GetGenericTypeDefinition()	Returns a Type object that represents a generic type definition from which the current generic type can be constructed.
GetHashCode()	Returns the hash code for this instance.
GetInterface(String)	Searches for the interface with the specified name.
GetInterface(String, Boolean)	When overridden in a derived class, searches for the specified interface, specifying whether to do a case-insensitive search for the interface name.
GetInterfaceMap(Type)	Returns an interface mapping for the specified interface type.
GetInterfaces()	When overridden in a derived class, gets all the interfaces implemented or inherited by the current Type.
GetMember(String)	Searches for the public members with the specified name.
GetMember(String, BindingFlags)	Searches for the specified members, using the specified binding constraints.
GetMember(String, MemberTypes, BindingFlags)	Searches for the specified members of the specified member type, using the specified binding constraints.
GetMembers()	Returns all the public members of the current Type.
GetMembers(BindingFlags)	When overridden in a derived class, searches for the members defined for the current Type, using the specified binding constraints.
GetMemberWithSameMetadataDefinition As(MemberInfo)	Searches for the MemberInfo on the current Type that matches the specified MemberInfo.

GetMethod(String)	Searches for the public method with the specified name.
GetMethod(String, BindingFlags)	Searches for the specified method, using the specified binding constraints.
GetMethod(String, BindingFlags, Binder, Calling Conventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetMethod(String, BindingFlags, Type[])	Searches for the specified method whose parameters match the specified argument types, using the specified binding constraints.
GetMethod(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethod(String, Int32, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints.
GetMethod(String, Int32, Type[])	Searches for the specified public method whose parameters match the specified generic parameter count and argument types.
GetMethod(String, Int32, Type[], Parameter Modifier[])	Searches for the specified public method whose parameters match the specified generic parameter count, argument types and modifiers.
GetMethod(String, Type[])	Searches for the specified public method whose parameters match the specified argument types.
GetMethod(String, Type[], ParameterModifier[])	Searches for the specified public method whose parameters match the specified

	argument types and modifiers.
GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethodImpl(String, Int32, BindingFlags, Binder, CallingConventions, Type[], Parameter Modifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethods()	Returns all the public methods of the current Type.
GetMethods(BindingFlags)	When overridden in a derived class, searches for the methods defined for the current Type, using the specified binding constraints.
GetNestedType(String)	Searches for the public nested type with the specified name.
GetNestedType(String, BindingFlags)	When overridden in a derived class, searches for the specified nested type, using the specified binding constraints.
GetNestedTypes()	Returns the public types nested in the current Type.
GetNestedTypes(BindingFlags)	When overridden in a derived class, searches for the types nested in the current Type, using the specified binding constraints.
GetOptionalCustomModifiers()	When overridden in a derived class, returns the optional custom modifiers of the current Type.
GetProperties()	Returns all the public properties of the current Type.
GetProperties(BindingFlags)	When overridden in a derived class, searches for the properties of the current Type, using the specified binding constraints.
GetProperty(String)	Searches for the public property with the specified name.

GetProperty(String, BindingFlags)	Searches for the specified property, using the specified binding constraints.
GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])	Searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetProperty(String, Type)	Searches for the public property with the specified name and return type.
GetProperty(String, Type, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetProperty(String, Type, Type[], Parameter Modifier[])	Searches for the specified public property whose parameters match the specified argument types and modifiers.
GetProperty(String, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetRequiredCustomModifiers()	When overridden in a derived class, returns the required custom modifiers of the current Type.
GetType()	Gets the current Type.
GetType(String)	Gets the Type with the specified name, performing a case-sensitive search.
GetType(String, Boolean)	Gets the Type with the specified name, performing a case-sensitive search and specifying whether to throw an exception if the type is not found.
GetType(String, Boolean, Boolean)	Gets the Type with the specified name, specifying whether to throw an exception if the type is not found and whether to perform a case-sensitive search.
GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>)	Gets the type with the specified name, optionally providing custom methods to resolve the assembly and the type.

GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean)	Gets the type with the specified name, specifying whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean)	Gets the type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
GetTypeArray(Object[])	Gets the types of the objects in the specified array.
GetTypeCode(Type)	Gets the underlying type code of the specified Type.
GetTypeCodeImpl()	Returns the underlying type code of this Type instance.
GetTypeFromCLSID(Guid)	Gets the type associated with the specified class identifier (CLSID).
GetTypeFromCLSID(Guid, Boolean)	Gets the type associated with the specified class identifier (CLSID), specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromCLSID(Guid, String)	Gets the type associated with the specified class identifier (CLSID) from the specified server.
GetTypeFromCLSID(Guid, String, Boolean)	Gets the type associated with the specified class identifier (CLSID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromHandle(RuntimeTypeHandle)	Gets the type referenced by the specified type handle.
GetTypeFromProgID(String)	Gets the type associated with the specified program identifier (ProgID), returning null if an error is encountered while loading the Type.
GetTypeFromProgID(String, Boolean)	Gets the type associated with the specified program identifier (ProgID), specifying

	whether to throw an exception if an error occurs while loading the type.
GetTypeFromProgID(String, String)	Gets the type associated with the specified program identifier (progID) from the specified server, returning null if an error is encountered while loading the type.
GetTypeFromProgID(String, String, Boolean)	Gets the type associated with the specified program identifier (progID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
GetTypeHandle(Object)	Gets the handle for the Type of a specified object.
HasElementTypeImpl()	When overridden in a derived class, implements the HasElementType property and determines whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
HasSameMetadataDefinitionAs(MemberInfo)	(Inherited from MemberInfo)
InvokeMember(String, BindingFlags, Binder, Object, Object[])	Invokes the specified member, using the specified binding constraints and matching the specified argument list.
InvokeMember(String, BindingFlags, Binder, Object, Object[], CultureInfo)	Invokes the specified member, using the specified binding constraints and matching the specified argument list and culture.
InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])	When overridden in a derived class, invokes the specified member, using the specified binding constraints and matching the specified argument list, modifiers and culture.
IsArrayImpl()	When overridden in a derived class, implements the IsArray property and determines whether the Type is an array.
IsAssignableFrom(Type)	Determines whether an instance of a specified type `c` can be assigned to a variable of the current type.
IsAssignableTo(Type)	Determines whether the current type can be assigned to a variable of the specified `targetType`.

IsByRefImpl()	When overridden in a derived class, implements the IsByRef property and determines whether the Type is passed by reference.
IsCOMObjectImpl()	When overridden in a derived class, implements the IsCOMObject property and determines whether the Type is a COM object.
IsContextfulImpl()	Implements the IsContextful property and determines whether the Type can be hosted in a context.
IsDefined(Type, Boolean)	When overridden in a derived class, indicates whether one or more attributes of the specified type or of its derived types is applied to this member. (Inherited from MemberInfo)
IsEnumDefined(Object)	Returns a value that indicates whether the specified value exists in the current enumeration type.
IsEquivalentTo(Type)	Determines whether two COM types have the same identity and are eligible for type equivalence.
IsInstanceOfType(Object)	Determines whether the specified object is an instance of the current Type.
IsMarshalByRefImpl()	Implements the IsMarshalByRef property and determines whether the Type is marshaled by reference.
IsPointerImpl()	When overridden in a derived class, implements the IsPointer property and determines whether the Type is a pointer.
IsPrimitiveImpl()	When overridden in a derived class, implements the IsPrimitive property and determines whether the Type is one of the primitive types.
IsSubclassOf(Type)	Determines whether the current Type derives from the specified Type.
IsValueTypeImpl()	Implements the IsValueType property and determines whether the Type is a value type; that is, not a class or an interface.

MakeArrayType()	Returns a Type object representing a one-dimensional array of the current type, with a lower bound of zero.
MakeArrayType(Int32)	Returns a Type object representing an array of the current type, with the specified number of dimensions.
MakeByRefType()	Returns a Type object that represents the current type when passed as a `ref` parameter (`ByRef` parameter in Visual Basic).
MakeGenericMethodParameter(Int32)	Returns a signature type object that can be passed into the `Type[]` array parameter of a GetMethod method to represent a generic parameter reference.
MakeGenericSignatureType(Type, Type[])	Creates a generic signature type, which allows third party reimplementations of Reflection to fully support the use of signature types in querying type members.
MakeGenericType(Type[])	Substitutes the elements of an array of types for the type parameters of the current generic type definition and returns a Type object representing the resulting constructed type.
MakePointerType()	Returns a Type object that represents a pointer to the current type.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ReflectionOnlyGetType(String, Boolean, Boolean)	**Obsolete.** Gets the Type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found. The type is loaded for reflection only, not for execution.
ToString()	Returns a `String` representing the name of the current `Type`.

Operators

Expand table

Equality(Type, Type)	Indicates whether two Type objects are equal.
Inequality(Type, Type)	Indicates whether two Type objects are not equal.

Extension Methods

Expand table

GetCustomAttribute(MemberInfo, Type)	Retrieves a custom attribute of a specified type that is applied to a specified member.
GetCustomAttribute(MemberInfo, Type, Boolean)	Retrieves a custom attribute of a specified type that is applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttribute<T>(MemberInfo)	Retrieves a custom attribute of a specified type that is applied to a specified member.
GetCustomAttribute<T>(MemberInfo, Boolean)	Retrieves a custom attribute of a specified type that is applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttributes(MemberInfo)	Retrieves a collection of custom attributes that are applied to a specified member.
GetCustomAttributes(MemberInfo, Boolean)	Retrieves a collection of custom attributes that are applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttributes(MemberInfo, Type)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member.
GetCustomAttributes(MemberInfo, Type, Boolean)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttributes<T>(MemberInfo)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member.
GetCustomAttributes<T>(MemberInfo, Boolean)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member, and optionally inspects the ancestors of that member.
IsDefined(MemberInfo, Type)	Indicates whether custom attributes of a specified type are applied to a specified member.
IsDefined(MemberInfo, Type, Boolean)	Indicates whether custom attributes of a specified type are applied to a specified member, and, optionally, applied to its ancestors.

GetTypeInfo(Type)	Returns the TypeInfo representation of the specified type.
GetMetadataToken(MemberInfo)	Gets a metadata token for the given member, if available.
HasMetadataToken(MemberInfo)	Returns a value that indicates whether a metadata token is available for the specified member.
GetRuntimeEvent(Type, String)	Retrieves an object that represents the specified event.
GetRuntimeEvents(Type)	Retrieves a collection that represents all the events defined on a specified type.
GetRuntimeField(Type, String)	Retrieves an object that represents a specified field.
GetRuntimeFields(Type)	Retrieves a collection that represents all the fields defined on a specified type.
GetRuntimeMethod(Type, String, Type[])	Retrieves an object that represents a specified method.
GetRuntimeMethods(Type)	Retrieves a collection that represents all methods defined on a specified type.
GetRuntimeProperties(Type)	Retrieves a collection that represents all the properties defined on a specified type.
GetRuntimeProperty(Type, String)	Retrieves an object that represents a specified property.
GetConstructor(Type, Type[])	
GetConstructors(Type)	
GetConstructors(Type, BindingFlags)	
GetDefaultMembers(Type)	
GetEvent(Type, String)	
GetEvent(Type, String, BindingFlags)	
GetEvents(Type)	
GetEvents(Type, BindingFlags)	
GetField(Type, String)	
GetField(Type, String, BindingFlags)	
GetFields(Type)	
GetFields(Type, BindingFlags)	

GetGenericArguments(Type)
GetInterfaces(Type)
GetMember(Type, String)
GetMember(Type, String, BindingFlags)
GetMembers(Type)
GetMembers(Type, BindingFlags)
GetMethod(Type, String)
GetMethod(Type, String, BindingFlags)
GetMethod(Type, String, Type[])
GetMethods(Type)
GetMethods(Type, BindingFlags)
GetNestedType(Type, String, BindingFlags)
GetNestedTypes(Type, BindingFlags)
GetProperties(Type)
GetProperties(Type, BindingFlags)
GetProperty(Type, String)
GetProperty(Type, String, BindingFlags)
GetProperty(Type, String, Type)
GetProperty(Type, String, Type, Type[])
IsAssignableFrom(Type, Type)
IsInstanceOfType(Type, Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Thread Safety

This type is thread safe.

See also

- Object
- System.Reflection
- ReflectionPermission
- Viewing Type Information

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Type class.

C#

```
protected Type ();
```

Remarks

This constructor is invoked by derived classes during the construction of type objects.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Delimiter Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Separates names in the namespace of the Type. This field is read-only.

C#

```
public static readonly char Delimiter;
```

Field Value

Char

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.EmptyTypes Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an empty array of type Type. This field is read-only.

C#

```
public static readonly Type[] EmptyTypes;
```

Field Value

Type[]

Examples

The following code example shows the `EmptyTypes` field used in one of the `GetConstructor` methods to get a constructor that takes no parameters.

C#

```
cInfo = type.GetConstructor (BindingFlags.ExactBinding, null,
         Type.EmptyTypes, null);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.FilterAttribute Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the member filter used on attributes. This field is read-only.

C#

```
public static readonly System.Reflection.MemberFilter FilterAttribute;
```

Field Value

MemberFilter

Examples

The following example gets the `FilterAttribute` delegate, passes it as a parameter to the FindMembers method, and displays the specified members and their attributes.

C#

```
using System;
using System.Reflection;
using System.Security;

public class MyFilterAttributeSample
{
    public static void Main()
    {
        try
        {
            MemberFilter myFilter = Type.FilterAttribute;
            Type myType = typeof(System.String);
            MemberInfo[] myMemberInfoArray =
myType.FindMembers(MemberTypes.Constructor
                |MemberTypes.Method, BindingFlags.Public |
BindingFlags.Static |
                BindingFlags.Instance, myFilter,
MethodAttributes.SpecialName);
            foreach (MemberInfo myMemberinfo in myMemberInfoArray)
            {
```

```
                Console.Write ("\n" + myMemberinfo.Name);
                Console.Write (" is a " +
myMemberinfo.MemberType.ToString());
            }
        }
        catch(ArgumentNullException e)
        {
            Console.Write("ArgumentNullException : " + e.Message);
        }
        catch(SecurityException e)
        {
            Console.Write("SecurityException : " + e.Message);
        }
        catch(Exception e)
        {
            Console.Write("Exception :" + e.Message);
        }
    }
}
```

Remarks

This field holds a reference to the delegate used by the FindMembers method. The method encapsulated by this delegate takes two parameters: the first is a MemberInfo object and the second is an `Object`. The method determines whether the `MemberInfo` object matches the criteria specified by the `Object`. The `Object` may be assigned the value of any one of the fields on the classes FieldAttributes, MethodAttributes, or MethodImplAttributes.

For example, the `Object` can be assigned the value of a field from `FieldAttributes` such as Public. In that case, when the `FilterAttribute` delegate is invoked, it will return `true` only if the method represented by the `MemberInfo` object is decorated with the public field attribute in metadata.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)
- MemberFilter

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.FilterName Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the case-sensitive member filter used on names. This field is read-only.

C#

```
public static readonly System.Reflection.MemberFilter FilterName;
```

Field Value

MemberFilter

Examples

The following code example gets the methods associated with the user-defined `Application` type.

C#

```
// Get the set of methods associated with the type
MemberInfo[] mi = typeof(Application).FindMembers(MemberTypes.Constructor |
    MemberTypes.Method,
    BindingFlags.Public | BindingFlags.Static | BindingFlags.NonPublic |
    BindingFlags.Instance | BindingFlags.DeclaredOnly,
    Type.FilterName, "*");
  Console.WriteLine("Number of methods (includes constructors): " +
mi.Length);
```

Remarks

This field holds a reference to the delegate used by the FindMembers method. The method encapsulated by this delegate takes two parameters: the first is a MemberInfo object and the second is an `Object`. The method determines whether the `MemberInfo` object matches the criteria specified by the `Object`. The `Object` is assigned a string

value, which may include a trailing "*" wildcard character. Only wildcard end string matching is supported.

For example, the `Object` may be assigned the value "Byte*". In that case, when the `FilterName` delegate is invoked, it will return `true` only if the method represented by the `MemberInfo` object has a name that begins with "Byte".

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)
- MemberFilter

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.FilterNameIgnoreCase Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the case-insensitive member filter used on names. This field is read-only.

C#

```
public static readonly System.Reflection.MemberFilter FilterNameIgnoreCase;
```

Field Value

MemberFilter

Examples

The following example gets the `MemberFilter` delegate, passes it as a parameter to the FindMembers method, and displays the methods and their attributes of the `String` class that begin with the letter "c", disregarding the case.

C#

```
using System;
using System.Reflection;
using System.Security;
public class MyFilterNameIgnoreCaseSample
{
    public static void Main()
    {
        try
        {
            MemberFilter myFilter = Type.FilterNameIgnoreCase;
            Type myType = typeof(System.String);
            MemberInfo[] myMemberinfo1 =
myType.FindMembers(MemberTypes.Constructor
                |MemberTypes.Method, BindingFlags.Public |
BindingFlags.Static |
                BindingFlags.Instance, myFilter, "C*");
            foreach (MemberInfo myMemberinfo2 in myMemberinfo1)
            {
```

```
                Console.Write("\n" + myMemberinfo2.Name);
                MemberTypes Mymembertypes = myMemberinfo2.MemberType;
                Console.WriteLine(" is a " + Mymembertypes.ToString());
            }
        }
        catch(ArgumentNullException e)
        {
            Console.Write("ArgumentNullException : " + e.Message);
        }
        catch(SecurityException e)
        {
            Console.Write("SecurityException : " + e.Message);
        }
        catch(Exception e)
        {
            Console.Write("Exception : " + e.Message);
        }
    }
}
```

Remarks

This field holds a reference to the delegate used by the FindMembers method. The method encapsulated by this delegate takes two parameters: the first is a MemberInfo object and the second is an `Object`. The method determines whether the `MemberInfo` object matches the criteria specified by the `Object`. The `Object` is assigned a string value, which may include a trailing "*" wildcard character. Only wildcard end string matching is supported.

For example, the `Object` may be assigned the value "ByTe*". In that case, when the `FilterName` delegate is invoked, it will return true only if the method represented by the `MemberInfo` object has a name that begins with "byte", ignoring case.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)
- MemberFilter

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Missing Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a missing value in the Type information. This field is read-only.

C#

```
public static readonly object Missing;
```

Field Value

Object

Examples

The following code example shows the use of the `Missing` field to invoke a method with its default arguments.

C#

```
using System;
using System.Reflection;
using System.CodeDom.Compiler;

class Example
{
    public static void Main()
    {
        // VB source for example. Not all versions of CS and CPP compilers
        // support optional arguments.
        string codeLines =
            "Imports System\n\n" +
            "Public Class OptionalArg\n" +
            "  Public Sub MyMethod(ByVal a As Integer, _\n" +
            "    Optional ByVal b As Double = 1.2, _\n" +
            "    Optional ByVal c As Integer = 1)\n\n" +
            "    Console.WriteLine(\"a = \" & a & \" b = \" & b & \" c = \"
& c)\n" +
            "  End Sub\n" +
            "End Class\n";
```

```
        // Generate a OptionalArg instance from the source above.
        object o = GenerateObjectFromSource("OptionalArg", codeLines,
"VisualBasic");
        Type t;

        t = o.GetType();
        BindingFlags bf = BindingFlags.Public | BindingFlags.Instance |
            BindingFlags.InvokeMethod | BindingFlags.OptionalParamBinding;

        t.InvokeMember("MyMethod", bf, null, o, new object[] {10, 55.3,
12});
        t.InvokeMember("MyMethod", bf, null, o, new object[] {10, 1.3,
Type.Missing});
        t.InvokeMember("MyMethod", bf, null, o, new object[] {10,
Type.Missing, Type.Missing});
    }

    private static object GenerateObjectFromSource(string objectName,
        string sourceLines, string providerName)
    {
        object genObject = null;
        CodeDomProvider codeProvider =
CodeDomProvider.CreateProvider(providerName);
        CompilerParameters cp = new CompilerParameters();

        cp.GenerateExecutable = false;
        cp.GenerateInMemory = true;

        CompilerResults results =
            codeProvider.CompileAssemblyFromSource(cp, sourceLines);
        if (results.Errors.Count == 0)
        {
            genObject = results.CompiledAssembly.CreateInstance(objectName);
        }

        return genObject;
    }
}
```

This code produces the following output:

a = 10 b = 55.3 c = 12

a = 10 b = 1.3 c = 1

a = 10 b = 1.2 c = 1

Remarks

Use the `Missing` field for invocation through reflection to obtain the default value of a parameter. If the `Missing` field is passed in for a parameter value and there is no default value for that parameter, an ArgumentException is thrown.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Missing

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Assembly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Assembly in which the type is declared. For generic types, gets the Assembly in which the generic type is defined.

C#

```
public abstract System.Reflection.Assembly Assembly { get; }
```

Property Value

Assembly

An Assembly instance that describes the assembly containing the current type. For generic types, the instance describes the assembly that contains the generic type definition, not the assembly that creates and uses a particular constructed type.

Examples

The following example displays the assembly name associated with the class and the fully qualified name of the type.

C#

```
using System;

class MyAssemblyClass
{
    public static void Main()
    {
        Type objType = typeof(Array);

        // Print the assembly full name.
        Console.WriteLine($"Assembly full name:\n
{objType.Assembly.FullName}.");

        // Print the assembly qualified name.
```

```
        Console.WriteLine($"Assembly qualified name:\n
{objType.AssemblyQualifiedName}.");
    }
}
// The example displays the following output if run under the .NET Framework
4.5:
//    Assembly full name:
//       mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089.
//    Assembly qualified name:
//       System.Array, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089.
```

Remarks

If the current Type object represents a constructed generic type, this property returns the assembly that contains the generic type definition. For example, suppose you create an assembly named MyGenerics.dll that contains the generic type definition `MyGenericStack<T>` (`MyGenericStack(Of T)` in Visual Basic, `generic<T> ref class MyGenericStack` in C++). If you create an instance of `MyGenericStack<int>` (`MyGenericStack(Of Integer)` in Visual Basic) in another assembly, the Assembly property for the constructed type returns an Assembly object that represents MyGenerics.dll.

Similarly, if the current Type object represents an unassigned generic parameter `T`, this property returns the assembly that contains the generic type that defines `T`.

If the Type.Assembly property is not available on a particular .NET implementation, such as .NET Core or the Universal Windows Platform, use the TypeInfo.Assembly property instead.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.AssemblyQualifiedName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the assembly-qualified name of the type, which includes the name of the assembly from which this Type object was loaded.

C#

```
public abstract string? AssemblyQualifiedName { get; }
```

Property Value

String

The assembly-qualified name of the Type, which includes the name of the assembly from which the Type was loaded, or `null` if the current instance represents a generic type parameter.

Examples

The following example displays the assembly name associated with the class and the fully qualified name of the type.

C#

```
using System;

class MyAssemblyClass
{
    public static void Main()
    {
        Type objType = typeof(Array);

        // Print the assembly full name.
        Console.WriteLine($"Assembly full name:\n
{objType.Assembly.FullName}.");

        // Print the assembly qualified name.
```

```
        Console.WriteLine($"Assembly qualified name:\n
{objType.AssemblyQualifiedName}.");
    }
}
// The example displays the following output if run under the .NET Framework
4.5:
//    Assembly full name:
//       mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089.
//    Assembly qualified name:
//       System.Array, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089.
```

The following example compares the strings returned by the ToString method and the Name, FullName, and `AssemblyQualifiedName` properties.

C#

```
using System;
using System.Collections.Generic;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        Type t = typeof(String);
        ShowTypeInfo(t);

        t = typeof(List<>);
        ShowTypeInfo(t);

        var list = new List<String>();
        t = list.GetType();
        ShowTypeInfo(t);

        Object v = 12;
        t = v.GetType();
        ShowTypeInfo(t);

        t = typeof(IFormatProvider);
        ShowTypeInfo(t);

        IFormatProvider ifmt = NumberFormatInfo.CurrentInfo;
        t = ifmt.GetType();
        ShowTypeInfo(t);
    }

    private static void ShowTypeInfo(Type t)
    {
        Console.WriteLine($"Name: {t.Name}");
        Console.WriteLine($"Full Name: {t.FullName}");
        Console.WriteLine($"ToString:  {t}");
```

```
        Console.WriteLine($"Assembly Qualified Name:
{t.AssemblyQualifiedName}");
        Console.WriteLine();
    }
}
// The example displays output like the following:
//    Name: String
//    Full Name: System.String
//    ToString:  System.String
//    Assembly Qualified Name: System.String, mscorlib, Version=4.0.0.0,
Culture=neutr
//    al, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1
//    ToString:  System.Collections.Generic.List`1[T]
//    Assembly Qualified Name: System.Collections.Generic.List`1, mscorlib,
Version=4.
//    0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1[[System.String, mscorlib,
Version=4
//    .0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]
//    ToString:  System.Collections.Generic.List`1[System.String]
//    Assembly Qualified Name:
System.Collections.Generic.List`1[[System.String, mscor
//    lib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089]], mscorl
//    ib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: Int32
//    Full Name: System.Int32
//    ToString:  System.Int32
//    Assembly Qualified Name: System.Int32, mscorlib, Version=4.0.0.0,
Culture=neutra
//    l, PublicKeyToken=b77a5c561934e089
//
//    Name: IFormatProvider
//    Full Name: System.IFormatProvider
//    ToString:  System.IFormatProvider
//    Assembly Qualified Name: System.IFormatProvider, mscorlib,
Version=4.0.0.0, Cult
//    ure=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: NumberFormatInfo
//    Full Name: System.Globalization.NumberFormatInfo
//    ToString:  System.Globalization.NumberFormatInfo
//    Assembly Qualified Name: System.Globalization.NumberFormatInfo,
mscorlib, Versio
//    n=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Remarks

The assembly-qualified name of a type consists of the type name, including its namespace, followed by a comma, followed by the display name of the assembly. The display name of an assembly is obtained using the Assembly.FullName property.

> **Note**
>
> In the .NET Framework version 2.0, processor architecture is added to assembly identity, and can be specified as part of assembly name strings. For example, "ProcessorArchitecture=msil". However, it is not included in the string returned by the **AssemblyQualifiedName** property, for compatibility reasons. See **AssemblyName.ProcessorArchitecture**.

All compilers that support the common language runtime emit the simple name of a nested class, and reflection constructs a mangled name when queried, in accordance with the following conventions.

Expand table

Delimiter	Meaning
Backslash (\)	Escape character.
Comma (,)	Precedes the Assembly name.
Plus sign (+)	Precedes a nested class.
Period (.)	Denotes namespace identifiers.
Brackets ([])	After a type name, denotes an array of that type. -or- For a generic type, encloses the generic type argument list. -or- Within a type argument list, encloses an assembly-qualified type.

For example, the assembly-qualified name for a class might look like this:

```
TopNamespace.SubNameSpace.ContainingClass+NestedClass, MyAssembly,
```

```
Version=1.3.0.0, Culture=neutral, PublicKeyToken=b17a5c561934e089
```

If the namespace contained a plus sign, for example TopNamespace.Sub+Namespace, then the plus sign (+) would be preceded by an escape character (\) to prevent it from being interpreted as a nesting separator. Reflection would emit this string as follows:

```

TopNamespace.Sub\+Namespace.ContainingClass+NestedClass, MyAssembly,
Version=1.3.0.0, Culture=neutral, PublicKeyToken=b17a5c561934e089
```

A "++" becomes "\+\+", and a "\" becomes "\\".

This qualified name can be persisted and later used to load the Type. To search for and load a Type, use GetType either with the type name only or with the assembly qualified type name. GetType with the type name only will look for the Type in the caller's assembly and then in the System assembly. GetType with the assembly qualified type name will look for the Type in any assembly.

Type names may include trailing characters that denote additional information about the type, such as whether the type is a reference type, a pointer type or an array type. To retrieve the type name without these trailing characters, use `t.GetElementType().ToString()`, where `t` is the type.

Spaces are relevant in all type name components except the assembly name. In the assembly name, spaces before the ',' separator are relevant, but spaces after the ',' separator are ignored.

Generic arguments of generic types are themselves qualified by assembly name. For example, in the assembly-qualified type name for `MyGenericClass<int>` (`MyGenericClass(Of Integer)` in Visual Basic), `int` is expanded to the assembly-qualified type name for Int32.

If the current Type object represents a generic parameter, this property returns `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String
- GetType()
- FullName
- Namespace
- AssemblyName
- Specifying Fully Qualified Type Names

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Attributes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the attributes associated with the Type.

C#

```
public System.Reflection.TypeAttributes Attributes { get; }
```

Property Value

TypeAttributes

A TypeAttributes object representing the attribute set of the Type, unless the Type represents a generic type parameter, in which case the value is unspecified.

Examples

The following example uses the Attributes property.

C#

```
using System;
using System.Reflection;

internal struct S
{
    public int X;
}

public abstract class Example
{
    protected sealed class NestedClass {}

    public interface INested {}

    public static void Main()
    {
        // Create an array of types.
```

```
        Type[] types = { typeof(Example), typeof(NestedClass),
                         typeof(INested), typeof(S) };

        foreach (var t in types)
        {
           Console.WriteLine("Attributes for type {0}:", t.Name);

           TypeAttributes attr = t.Attributes;

           // To test for visibility attributes, you must use the visibility
mask.
           TypeAttributes visibility = attr & TypeAttributes.VisibilityMask;
           switch (visibility)
           {
               case TypeAttributes.NotPublic:
                   Console.WriteLine("   ...is not public");
                   break;
               case TypeAttributes.Public:
                   Console.WriteLine("   ...is public");
                   break;
               case TypeAttributes.NestedPublic:
                   Console.WriteLine("   ...is nested and public");
                   break;
               case TypeAttributes.NestedPrivate:
                   Console.WriteLine("   ...is nested and private");
                   break;
               case TypeAttributes.NestedFamANDAssem:
                   Console.WriteLine("   ...is nested, and inheritable only
within the assembly" +
                       "\n         (cannot be declared in C#)");
                   break;
               case TypeAttributes.NestedAssembly:
                   Console.WriteLine("   ...is nested and internal");
                   break;
               case TypeAttributes.NestedFamily:
                   Console.WriteLine("   ...is nested and protected");
                   break;
               case TypeAttributes.NestedFamORAssem:
                   Console.WriteLine("   ...is nested and protected
internal");
                   break;
           }

           // Use the layout mask to test for layout attributes.
           TypeAttributes layout = attr & TypeAttributes.LayoutMask;
           switch (layout)
           {
               case TypeAttributes.AutoLayout:
                   Console.WriteLine("   ...is AutoLayout");
                   break;
               case TypeAttributes.SequentialLayout:
                   Console.WriteLine("   ...is SequentialLayout");
                   break;
               case TypeAttributes.ExplicitLayout:
                   Console.WriteLine("   ...is ExplicitLayout");
```

```
                    break;
            }

            // Use the class semantics mask to test for class semantics
attributes.
            TypeAttributes classSemantics = attr &
TypeAttributes.ClassSemanticsMask;
            switch (classSemantics)
            {
                case TypeAttributes.Class:
                    if (t.IsValueType)
                    {
                        Console.WriteLine("   ...is a value type");
                    }
                    else
                    {
                        Console.WriteLine("   ...is a class");
                    }
                    break;
                case TypeAttributes.Interface:
                    Console.WriteLine("   ...is an interface");
                    break;
            }

            if ((attr & TypeAttributes.Abstract) != 0)
            {
                Console.WriteLine("   ...is abstract");
            }

            if ((attr & TypeAttributes.Sealed) != 0)
            {
                Console.WriteLine("   ...is sealed");
            }

            Console.WriteLine();
        }
    }
}
// The example displays the following output:
// Attributes for type Example:
//    ...is public
//    ...is AutoLayout
//    ...is a class
//    ...is abstract

// Attributes for type NestedClass:
//    ...is nested and protected
//    ...is AutoLayout
//    ...is a class
//    ...is sealed

// Attributes for type INested:
//    ...is nested and public
//    ...is AutoLayout
//    ...is an interface
```

```
//    ...is abstract

// Attributes for type S:
//    ...is not public
//    ...is SequentialLayout
//    ...is a value type
//    ...is sealed
```

Remarks

Some members of the TypeAttributes enumeration are masks that represent a group of values. Each group includes one member whose underlying value is zero. For example, the underlying value of the TypeAttributes.NotPublic member in the TypeAttributes.VisibilityMask group is zero, as is the TypeAttributes.AutoLayout member in the TypeAttributes.SequentialLayout group. Because of this, you must use the mask before testing for those values. The example provides an illustration.

> **Tip**
>
> For most purposes, properties like **IsClass**,**IsAutoLayout**, and **IsSpecialName** are easier to use than type attributes.

If the current Type represents a constructed generic type, this property returns the attributes of the generic type definition. For example, the attributes returned for `MyGenericClass<int>` (`MyGenericClass(Of Integer)` in Visual Basic) are the attributes of `MyGenericClass<T>` (`MyGenericClass(Of T)` in Visual Basic).

If the current Type represents a generic type parameter - that is, if the IsGenericParameter property returns `true` - the TypeAttributes value returned by this property is unspecified.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetAttributeFlagsImpl()
- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.BaseType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type from which the current Type directly inherits.

C#

```
public abstract Type? BaseType { get; }
```

Property Value

Type

The Type from which the current Type directly inherits, or `null` if the current `Type` represents the Object class or an interface.

Examples

The following example demonstrates using the BaseType property.

C#

```
using System;
class TestType
{
    public static void Main()
    {
        Type t = typeof(int);
        Console.WriteLine("{0} inherits from {1}.", t,t.BaseType);
    }
}
```

The following example uses recursion to list the complete inheritance hierarchy of each class found in an assembly. The example defines a class named `C` that derives from a class named `B`, which, in turn, derives from a class named `A`.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      foreach (var t in typeof(Example).Assembly.GetTypes()) {
         Console.WriteLine("{0} derived from: ", t.FullName);
         var derived = t;
         do {
            derived = derived.BaseType;
            if (derived != null)
               Console.WriteLine("   {0}", derived.FullName);
         } while (derived != null);
         Console.WriteLine();
      }
   }
}

public class A {}

public class B : A
{}

public class C : B
{}
// The example displays the following output:
//       Example derived from:
//          System.Object
//
//       A derived from:
//          System.Object
//
//       B derived from:
//          A
//          System.Object
//
//       C derived from:
//          B
//          A
//          System.Object
```

Remarks

The base type is the type from which the current type directly inherits. Object is the only type that does not have a base type, therefore `null` is returned as the base type of Object.

Interfaces inherit from zero or more base interfaces; therefore, this property returns `null` if the `Type` object represents an interface. The base interfaces can be determined

with GetInterfaces or FindInterfaces.

If the current Type represents a constructed generic type, the base type reflects the generic arguments. For example, consider the following declarations:

```
C#

class B<U> { }
class C<T> : B<T> { }
```

For the constructed type `C<int>` (`C(Of Integer)` in Visual Basic), the BaseType property returns `B<int>`.

If the current Type represents a type parameter of a generic type definition, BaseType returns the class constraint, that is, the class the type parameter must inherit. If there is no class constraint, BaseType returns System.Object.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Object
- GetInterfaces()
- FindInterfaces(TypeFilter, Object)
- UnderlyingSystemType
- IsSubclassOf(Type)
- Reflection and Generic Types

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.ContainsGenericParameters Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type object has type parameters that have not been replaced by specific types.

C#

```
public virtual bool ContainsGenericParameters { get; }
```

Property Value

Boolean

`true` if the Type object is itself a generic type parameter or has type parameters for which specific types have not been supplied; otherwise, `false`.

Examples

The following example defines a generic class with two type parameters and then defines a second generic class that derives from the first class. The derived class's base class has two type arguments: the first is Int32 and the second is a type parameter of the derived type. The example displays information about these generic classes, including the positions reported by the GenericParameterPosition property.

C#

```
using System;
using System.Reflection;
using System.Collections.Generic;

// Define a base class with two type parameters.
public class Base<T, U> { }

// Define a derived class. The derived class inherits from a constructed
```

```
// class that meets the following criteria:
//   (1) Its generic type definition is Base<T, U>.
//   (2) It specifies int for the first type parameter.
//   (3) For the second type parameter, it uses the same type that is used
//       for the type parameter of the derived class.
// Thus, the derived class is a generic type with one type parameter, but
// its base class is an open constructed type with one type argument and
// one type parameter.
public class Derived<V> : Base<int, V> { }

public class Test
{
    public static void Main()
    {
        Console.WriteLine(
            "\r\n--- Display a generic type and the open constructed");
        Console.WriteLine("    type from which it is derived.");

        // Create a Type object representing the generic type definition
        // for the Derived type, by omitting the type argument. (For
        // types with multiple type parameters, supply the commas but
        // omit the type arguments.)
        //
        Type derivedType = typeof(Derived<>);
        DisplayGenericTypeInfo(derivedType);

        // Display its open constructed base type.
        DisplayGenericTypeInfo(derivedType.BaseType);
    }

    private static void DisplayGenericTypeInfo(Type t)
    {
        Console.WriteLine("\r\n{0}", t);

        Console.WriteLine("\tIs this a generic type definition? {0}",
            t.IsGenericTypeDefinition);

        Console.WriteLine("\tIs it a generic type? {0}",
            t.IsGenericType);

        Console.WriteLine("\tDoes it have unassigned generic parameters?
{0}",
            t.ContainsGenericParameters);

        if (t.IsGenericType)
        {
            // If this is a generic type, display the type arguments.
            //
            Type[] typeArguments = t.GetGenericArguments();

            Console.WriteLine("\tList type arguments ({0}):",
                typeArguments.Length);

            foreach (Type tParam in typeArguments)
            {
```

```
                // IsGenericParameter is true only for generic type
                // parameters.
                //
                if (tParam.IsGenericParameter)
                {
                    Console.WriteLine(
                        "\t\t{0}  (unassigned - parameter position {1})",
                        tParam,
                        tParam.GenericParameterPosition);
                }
                else
                {
                    Console.WriteLine("\t\t{0}", tParam);
                }
            }
        }
    }
}

/* This example produces the following output:

--- Display a generic type and the open constructed
    type from which it is derived.

Derived`1[V]
        Is this a generic type definition? True
        Is it a generic type? True
        Does it have unassigned generic parameters? True
        List type arguments (1):
                V  (unassigned - parameter position 0)

Base`2[System.Int32,V]
        Is this a generic type definition? False
        Is it a generic type? True
        Does it have unassigned generic parameters? True
        List type arguments (2):
                System.Int32
                V  (unassigned - parameter position 0)
 */
```

Remarks

In order to create an instance of a type, there must be no generic type definitions or open constructed types in the type arguments of the type itself, in any enclosing generic types, or in any elements of the type. Another way of saying this is that when examined recursively, the type must contain no generic type parameters.

Since types can be arbitrarily complex, making this determination is difficult. For convenience and to reduce the chance of error, the ContainsGenericParameters property provides a standard way to distinguish between closed constructed types, which can be

instantiated, and open constructed types, which cannot. If the ContainsGenericParameters property returns `true`, the type cannot be instantiated.

The ContainsGenericParameters property searches recursively for type parameters. For example, it returns `true` for an array whose elements are type `A<T>` (`A(Of T)` in Visual Basic), even though the array is not itself generic. Contrast this with the behavior of the IsGenericType property, which returns `false` for arrays.

For a set of example classes and a table showing the values of the ContainsGenericParameters property, see IsGenericType.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ContainsGenericParameters
- IsGenericTypeDefinition
- MakeGenericType(Type[])
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.DeclaringMethod Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a MethodBase that represents the declaring method, if the current Type represents a type parameter of a generic method.

C#

```
public virtual System.Reflection.MethodBase? DeclaringMethod { get; }
```

Property Value

MethodBase

If the current Type represents a type parameter of a generic method, a MethodBase that represents declaring method; otherwise, `null`.

Examples

The following code example defines a class that has a generic method, assigns a type argument to the method, and invokes the resulting constructed generic method. It also displays information about the generic method definition and the constructed method. When displaying information about the type parameters of the generic method definition, in the `DisplayGenericMethodInfo` method, the example code shows the value of the DeclaringMethod property for the method's generic type parameter.

C#

```
using System;
using System.Reflection;

// Define a class with a generic method.
public class Example
{
    public static void Generic<T>(T toDisplay)
    {
        Console.WriteLine("\r\nHere it is: {0}", toDisplay);
```

```
    }
}

public class Test
{
    public static void Main()
    {
        Console.WriteLine("\r\n--- Examine a generic method.");

        // Create a Type object representing class Example, and
        // get a MethodInfo representing the generic method.
        //
        Type ex = typeof(Example);
        MethodInfo mi = ex.GetMethod("Generic");

        DisplayGenericMethodInfo(mi);

        // Assign the int type to the type parameter of the Example
        // method.
        //
        MethodInfo miConstructed = mi.MakeGenericMethod(typeof(int));

        DisplayGenericMethodInfo(miConstructed);

        // Invoke the method.
        object[] args = {42};
        miConstructed.Invoke(null, args);

        // Invoke the method normally.
        Example.Generic<int>(42);

        // Get the generic type definition from the closed method,
        // and show it's the same as the original definition.
        //
        MethodInfo miDef = miConstructed.GetGenericMethodDefinition();
        Console.WriteLine("\r\nThe definition is the same: {0}",
            miDef == mi);
    }

    private static void DisplayGenericMethodInfo(MethodInfo mi)
    {
        Console.WriteLine("\r\n{0}", mi);

        Console.WriteLine("\tIs this a generic method definition? {0}",
            mi.IsGenericMethodDefinition);

        Console.WriteLine("\tIs it a generic method? {0}",
            mi.IsGenericMethod);

        Console.WriteLine("\tDoes it have unassigned generic parameters?
{0}",
            mi.ContainsGenericParameters);

        // If this is a generic method, display its type arguments.
        //
```

```
        if (mi.IsGenericMethod)
        {
            Type[] typeArguments = mi.GetGenericArguments();

            Console.WriteLine("\tList type arguments ({0}):",
                typeArguments.Length);

            foreach (Type tParam in typeArguments)
            {
                // IsGenericParameter is true only for generic type
                // parameters.
                //
                if (tParam.IsGenericParameter)
                {
                    Console.WriteLine("\t\t{0}  parameter position {1}" +
                        "\n\t\t   declaring method: {2}",
                        tParam,
                        tParam.GenericParameterPosition,
                        tParam.DeclaringMethod);
                }
                else
                {
                    Console.WriteLine("\t\t{0}", tParam);
                }
            }
        }
    }
}

/* This example produces the following output:

--- Examine a generic method.

Void Generic[T](T)
        Is this a generic method definition? True
        Is it a generic method? True
        Does it have unassigned generic parameters? True
        List type arguments (1):
                T  parameter position 0
                   declaring method: Void Generic[T](T)

Void Generic[Int32](Int32)
        Is this a generic method definition? False
        Is it a generic method? True
        Does it have unassigned generic parameters? False
        List type arguments (1):
                System.Int32

Here it is: 42

Here it is: 42

The definition is the same: True

 */
```

Remarks

The declaring method is a generic method definition. That is, if DeclaringMethod does not return `null`, then `DeclaringMethod.IsGenericMethodDefinition` returns `true`.

The DeclaringType and DeclaringMethod properties identify the generic type definition or generic method definition in which the generic type parameter was originally defined:

- If the DeclaringMethod property returns a MethodInfo, that MethodInfo represents a generic method definition, and the current Type object represents a type parameter of that generic method definition.
- If the DeclaringMethod property returns `null`, then the DeclaringType property always returns a Type object representing a generic type definition, and the current Type object represents a type parameter of that generic type definition.
- Getting the DeclaringMethod property on a type whose IsGenericParameter property is `false` throws an InvalidOperationException.

The MethodBase that is returned by the DeclaringMethod property is either a MethodInfo in the case of a generic method, or a ConstructorInfo in the case of a generic constructor.

> **Note**
>
> In the .NET Framework version 2.0, generic constructors are not supported.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsGenericParameter
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.DeclaringType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type that declares the current nested type or generic type parameter.

C#

```
public override Type? DeclaringType { get; }
```

Property Value

Type

A Type object representing the enclosing type, if the current type is a nested type; or the generic type definition, if the current type is a type parameter of a generic type; or the type that declares the generic method, if the current type is a type parameter of a generic method; otherwise, `null`.

Examples

This example displays the declaring type of a method in a derived class.

C#

```
using System;
using System.Reflection;

public abstract class dtype
{

    public abstract class MyClassA
    {
        public abstract int m();
    }

    public abstract class MyClassB : MyClassA
    {
    }
```

```
    public static void Main(string[] args)
    {
        Console.WriteLine("The declaring type of m is {0}.",
            typeof(MyClassB).GetMethod("m").DeclaringType);
    }
}
/* The example produces the following output:

The declaring type of m is dtype+MyClassA.
*/
```

Remarks

If the current Type object represents a type parameter of a generic type, this property returns the generic type definition.

If the current Type object represents a type parameter of a generic method, this property returns the type that contains the generic method definition. If the type is generic, the generic type definition is returned. That is, the following code returns the generic type definition of the List<T> generic class, which contains the ConvertAll generic method:

```
C#

Type t = typeof(List<string>).GetMethod("ConvertAll").GetGenericArguments()
[0].DeclaringType;
```

If the current Type represents a type parameter in the definition of a generic type or generic method, the DeclaringType and DeclaringMethod properties identify the generic type definition or generic method definition where the generic type parameter was originally defined:

- If the DeclaringMethod property returns a MethodInfo, that MethodInfo represents a generic method definition, and the current Type object represents a type parameter of that generic method definition.
- If the DeclaringMethod property returns `null`, then the DeclaringType property always returns a Type object representing a generic type definition, and the current Type object represents a type parameter of that generic type definition.
- Getting the DeclaringType property on a type whose IsGenericParameter property is `false` throws an InvalidOperationException.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MemberInfo
- ReflectedType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.DefaultBinder Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a reference to the default binder, which implements internal rules for selecting the appropriate members to be called by InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[]).

C#

```
public static System.Reflection.Binder DefaultBinder { get; }
```

Property Value

Binder

A reference to the default binder used by the system.

Examples

The following example gets the default binder from the `DefaultBinder` property, and invokes a member of MyClass by passing the `DefaultBinder` value as a parameter to InvokeMember.

C#

```
using System;
using System.Reflection;

public class MyDefaultBinderSample
{
    public static void Main()
    {
        try
        {
            Binder defaultBinder = Type.DefaultBinder;
            MyClass myClass = new MyClass();
            // Invoke the HelloWorld method of MyClass.
            myClass.GetType().InvokeMember("HelloWorld",
```

```
BindingFlags.InvokeMethod,
                defaultBinder, myClass, new object [] {});
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception :" + e.Message);
        }
    }

    class MyClass
    {
        public void HelloWorld()
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Remarks

The default binder provided with the common language runtime is applicable in all but the most specialized circumstances. If you need a binder that follows rules that differ from those of the supplied default binder, define a type derived from the Binder class and pass an instance of that type using the `binder` parameter of one of the InvokeMember overloads.

Reflection models the accessibility rules of the common type system. For example, if the caller is in the same assembly, the caller does not need special permissions for internal members. Otherwise, the caller needs ReflectionPermission. This is consistent with lookup of members that are protected, private, and so on.

The general principle is that ChangeType should perform only widening conversions, which never lose data. An example of a widening conversion is converting a value that is a 32-bit signed integer to a value that is a 64-bit signed integer. This is distinguished from a narrowing conversion, which may lose data. An example of a narrowing conversion is converting a 64-bit signed integer to a 32-bit signed integer.

The following table lists the conversions supported by the default binder.

Expand table

Source Type	Target Type
Any type	Its base type.
Any type	The interface it implements.

Source Type	Target Type
Char	Unt16, UInt32, Int32, UInt64, Int64, Single, Double
Byte	Char, Unt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double
SByte	Int16, Int32, Int64, Single, Double
UInt16	UInt32, Int32, UInt64, Int64, Single, Double
Int16	Int32, Int64, Single, Double
UInt32	UInt64, Int64, Single, Double
Int32	Int64, Single, Double
UInt64	Single, Double
Int64	Single, Double
Single	Double
Non-reference	By-reference.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Binder

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Type.FullName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fully qualified name of the type, including its namespace but not its assembly.

C#

```
public abstract string? FullName { get; }
```

Property Value

String

The fully qualified name of the type, including its namespace but not its assembly; or `null` if the current instance represents a generic type parameter, an array type, pointer type, or `byref` type based on a type parameter, or a generic type that is not a generic type definition but contains unresolved type parameters.

Examples

The following example displays the full name of the specified type.

C#

```
using System;
class TestFullName
{
public static void Main()
    {
    Type t = typeof(Array);
    Console.WriteLine("The full name of the Array type is {0}.",
t.FullName);
    }
}

/* This example produces the following output:
```

```
The full name of the Array type is System.Array.
 */
```

The following example compares the strings returned by the ToString method and the Name, `FullName`, and AssemblyQualifiedName properties.

C#

```
using System;
using System.Collections.Generic;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        Type t = typeof(String);
        ShowTypeInfo(t);

        t = typeof(List<>);
        ShowTypeInfo(t);

        var list = new List<String>();
        t = list.GetType();
        ShowTypeInfo(t);

        Object v = 12;
        t = v.GetType();
        ShowTypeInfo(t);

        t = typeof(IFormatProvider);
        ShowTypeInfo(t);

        IFormatProvider ifmt = NumberFormatInfo.CurrentInfo;
        t = ifmt.GetType();
        ShowTypeInfo(t);
    }

    private static void ShowTypeInfo(Type t)
    {
        Console.WriteLine($"Name: {t.Name}");
        Console.WriteLine($"Full Name: {t.FullName}");
        Console.WriteLine($"ToString:  {t}");
        Console.WriteLine($"Assembly Qualified Name:
{t.AssemblyQualifiedName}");
        Console.WriteLine();
    }
}
// The example displays output like the following:
//    Name: String
//    Full Name: System.String
//    ToString:  System.String
//    Assembly Qualified Name: System.String, mscorlib, Version=4.0.0.0,
```

```
Culture=neutr
//    al, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1
//    ToString:  System.Collections.Generic.List`1[T]
//    Assembly Qualified Name: System.Collections.Generic.List`1, mscorlib,
Version=4.
//    0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1[[System.String, mscorlib,
Version=4
//    .0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]
//    ToString:  System.Collections.Generic.List`1[System.String]
//    Assembly Qualified Name:
System.Collections.Generic.List`1[[System.String, mscor
//    lib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089]], mscorl
//    ib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: Int32
//    Full Name: System.Int32
//    ToString:  System.Int32
//    Assembly Qualified Name: System.Int32, mscorlib, Version=4.0.0.0,
Culture=neutra
//    l, PublicKeyToken=b77a5c561934e089
//
//    Name: IFormatProvider
//    Full Name: System.IFormatProvider
//    ToString:  System.IFormatProvider
//    Assembly Qualified Name: System.IFormatProvider, mscorlib,
Version=4.0.0.0, Cult
//    ure=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: NumberFormatInfo
//    Full Name: System.Globalization.NumberFormatInfo
//    ToString:  System.Globalization.NumberFormatInfo
//    Assembly Qualified Name: System.Globalization.NumberFormatInfo,
mscorlib, Versio
//    n=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Remarks

For example, the fully qualified name of the String type is `System.String`. Contrast this with the assembly-qualified name returned by the AssemblyQualifiedName property, which consists of the full name plus the full assembly name.

If the current type represents a closed generic type, the type arguments in the string returned by the FullName property are qualified by their full assembly name, even

though the string representation of the generic type itself is not qualified by its full assembly name. The following example illustrates the difference in the FullName property for a type that represents generic type definition and one that represents a closed generic type.

C#

```
using System;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Type t = typeof(List<>);
      Console.WriteLine(t.FullName);
      Console.WriteLine();

      List<String> list = new List<String>();
      t = list.GetType();
      Console.WriteLine(t.FullName);
   }
}
// The example displays the following output:
// System.Collections.Generic.List`1
//
// System.Collections.Generic.List`1[[System.String, mscorlib,
//        Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089]]
```

This property returns `null` if:

- The current Type object represents a type parameter of a generic type.

  The following example retrieves the type parameter of the Nullable<T> type and attempts to display its FullName property.

  C#

  ```
  using System;
  using System.Reflection;

  public class Example
  {
     public static void Main()
     {
        Type t = typeof(Nullable<>);
        Console.WriteLine(t.FullName);
        if (t.IsGenericType) {
           Console.Write("   Generic Type Parameters: ");
           Type[] gtArgs = t.GetGenericArguments();
  ```

```
            for (int ctr = 0; ctr < gtArgs.Length; ctr++) {
               Console.WriteLine(gtArgs[ctr].FullName ??
                                 "(unassigned) " +
gtArgs[ctr].ToString());
             }
            Console.WriteLine();
         }
      }
}
// The example displays the following output:
//       System.Nullable`1
//          Generic Type Parameters: (unassigned) T
```

- The current Type object represents an array type, a pointer type, or a `byref` type that is based on a generic type parameter.

  The following example defines a generic type, `Genericype1<T>`, with three methods: `Display(T[])`, which is passed an array of type T; `HandleT(T)`, which is passed a T object; and `ChangeValue(ref T)`, which is passed a T object by reference. Because C# and Visual Basic do not allow us to define T as a pointer in the `HandleT` method, we have to call the MakePointerType method on the Type object that represents the method's parameter type to create a pointer to a generic type. The output from the example shows that in all three cases, the FullName property is `null`.

C#

```
using System;
using System.Reflection;

public class GenericType1<T>
{
   public void Display(T[] elements)
   {}

   public void HandleT(T obj)
   {}

   public bool ChangeValue(ref T arg)
   {
      return true;
   }
}

public class Example
{
   public static void Main()
   {
      Type t = typeof(GenericType1<>);
      Console.WriteLine("Type Name: {0}", t.FullName);
```

```
      MethodInfo[] methods = t.GetMethods(BindingFlags.Instance |
                                          BindingFlags.DeclaredOnly |
                                          BindingFlags.Public);
      foreach (var method in methods) {
         Console.WriteLine("   Method: {0}", method.Name);
         // Get method parameters.
         ParameterInfo param = method.GetParameters()[0];
         Type paramType = param.ParameterType;
         if (method.Name == "HandleT")
            paramType = paramType.MakePointerType();
         Console.WriteLine("      Parameter: {0}",
                           paramType.FullName ??
                           paramType.ToString() + " (unassigned)");
      }
   }
}
// The example displays the following output:
//       Type Name: GenericType1`1
//          Method: Display
//             Parameter: T[] (unassigned))
//          Method: HandleT
//             Parameter: T* (unassigned)
//          Method: ChangeValue
//             Parameter: T& (unassigned)
```

- The current type contains generic type parameters that have not been replaced by specific types (that is, the ContainsGenericParameters property returns `true`), but the type is not a generic type definition (that is, the IsGenericTypeDefinition property returns `false`

  In the following example, `Derived<T>` inherits from `Base<T>`. The BaseType property obtains the Type object that represents the base type of `Derived<T>`, and its FullName property returns `null`.

C#

```
using System;
using System.Reflection;

public class Base<T> { }

public class Derived<T> : Base<T> { }

public class Example
{
   public static void Main()
   {
      Type t = typeof(Derived<>);
      Console.WriteLine("Generic Class: {0}", t.FullName);
      Console.WriteLine("   Contains Generic Paramters: {0}",
                        t.ContainsGenericParameters);
```

```
        Console.WriteLine("   Generic Type Definition: {0}\n",
                          t.IsGenericTypeDefinition);

        Type baseType = t.BaseType;
        Console.WriteLine("Its Base Class: {0}",
                          baseType.FullName ??
                          "(unassigned) " + baseType.ToString());
        Console.WriteLine("   Contains Generic Paramters: {0}",
                          baseType.ContainsGenericParameters);
        Console.WriteLine("   Generic Type Definition: {0}",
                          baseType.IsGenericTypeDefinition);
        Console.WriteLine("   Full Name: {0}\n",
                          baseType.GetGenericTypeDefinition().FullName);

        t = typeof(Base<>);
        Console.WriteLine("Generic Class: {0}", t.FullName);
        Console.WriteLine("   Contains Generic Paramters: {0}",
                          t.ContainsGenericParameters);
        Console.WriteLine("   Generic Type Definition: {0}\n",
                          t.IsGenericTypeDefinition);
    }
}
// The example displays the following output:
//       Generic Class: Derived`1
//          Contains Generic Paramters: True
//          Generic Type Definition: True
//
//       Its Base Class: (unassigned) Base`1[T]
//          Contains Generic Paramters: True
//          Generic Type Definition: False
//          Full Name: Base`1
//
//       Generic Class: Base`1
//          Contains Generic Paramters: True
//          Generic Type Definition: True
```

To get a FullName that is not `null`, you can use the GetGenericTypeDefinition method to get the generic type definition, as the example illustrates.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- String
- Namespace
- AssemblyQualifiedName
- AssemblyName
- Specifying Fully Qualified Type Names

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GenericParameterAttributes Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a combination of GenericParameterAttributes flags that describe the covariance and special constraints of the current generic type parameter.

C#

```
public virtual System.Reflection.GenericParameterAttributes
GenericParameterAttributes { get; }
```

Property Value

GenericParameterAttributes

A bitwise combination of GenericParameterAttributes values that describes the covariance and special constraints of the current generic type parameter.

Exceptions

InvalidOperationException

The current Type object is not a generic type parameter. That is, the IsGenericParameter property returns `false`.

NotSupportedException

The invoked method is not supported in the base class.

Examples

The following code example defines a generic type `Test` with two type parameters that have different constraints. When the program executes, the constraints are examined using the GenericParameterAttributes property and the GetGenericParameterConstraints method.

C#

```
using System;
using System.Reflection;

// Define a sample interface to use as an interface constraint.
public interface ITest {}

// Define a base type to use as a base class constraint.
public class Base {}

// Define the generic type to examine. The first generic type parameter,
// T, derives from the class Base and implements ITest. This demonstrates
// a base class constraint and an interface constraint. The second generic
// type parameter, U, must be a reference type (class) and must have a
// default constructor (new()). This demonstrates special constraints.
//
public class Test<T,U>
    where T : Base, ITest
    where U : class, new() {}

// Define a type that derives from Base and implements ITest. This type
// satisfies the constraints on T in class Test.
public class Derived : Base, ITest {}

public class Example
{
    public static void Main()
    {
        // To get the generic type definition, omit the type
        // arguments but retain the comma to indicate the number
        // of type arguments.
        //
        Type def = typeof(Test<,>);
        Console.WriteLine("\r\nExamining generic type {0}", def);

        // Get the type parameters of the generic type definition,
        // and display them.
        //
        Type[] defparams = def.GetGenericArguments();
        foreach (Type tp in defparams)
        {
            Console.WriteLine("\r\nType parameter: {0}", tp.Name);
            Console.WriteLine("\t{0}",
                ListGenericParameterAttributes(tp));

            // List the base class and interface constraints. The
            // constraints are returned in no particular order. If
            // there are no class or interface constraints, an empty
            // array is returned.
            //
            Type[] tpConstraints = tp.GetGenericParameterConstraints();
            foreach (Type tpc in tpConstraints)
            {
```

```
                Console.WriteLine("\t{0}", tpc);
            }
        }
    }

    // List the variance and special constraint flags.
    //
    private static string ListGenericParameterAttributes(Type t)
    {
        string retval;
        GenericParameterAttributes gpa = t.GenericParameterAttributes;
        GenericParameterAttributes variance = gpa &
            GenericParameterAttributes.VarianceMask;

        // Select the variance flags.
        if (variance == GenericParameterAttributes.None)
        {
            retval = "No variance flag;";
        }
        else
        {
            if ((variance & GenericParameterAttributes.Covariant) != 0)
                retval = "Covariant;";
            else
                retval = "Contravariant;";
        }

        // Select
        GenericParameterAttributes constraints = gpa &
            GenericParameterAttributes.SpecialConstraintMask;

        if (constraints == GenericParameterAttributes.None)
        {
            retval += " No special constraints";
        }
        else
        {
            if ((constraints &
GenericParameterAttributes.ReferenceTypeConstraint) != 0)
                retval += " ReferenceTypeConstraint";
            if ((constraints &
GenericParameterAttributes.NotNullableValueTypeConstraint) != 0)
                retval += " NotNullableValueTypeConstraint";
            if ((constraints &
GenericParameterAttributes.DefaultConstructorConstraint) != 0)
                retval += " DefaultConstructorConstraint";
        }

        return retval;
    }
}
/* This example produces the following output:

Examining generic type Test`2[T,U]
```

```
Type parameter: T
        No variance flag; no special constraints.
        Base
        ITest

Type parameter: U
        No variance flag; ReferenceTypeConstraint
DefaultConstructorConstraint
 */
```

Remarks

The value of this property contains flags that describe whether the current generic type parameter is covariant, and flags that describe any special constraints. Use the GenericParameterAttributes.VarianceMask value to select the covariance flags, and use the GenericParameterAttributes.SpecialConstraintMask value to select the constraint flags.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetGenericParameterConstraints()
- IsGenericParameter
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GenericParameterPosition Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the position of the type parameter in the type parameter list of the generic type or method that declared the parameter, when the Type object represents a type parameter of a generic type or a generic method.

```
C#

public virtual int GenericParameterPosition { get; }
```

Property Value

Int32

The position of a type parameter in the type parameter list of the generic type or method that defines the parameter. Position numbers begin at 0.

Exceptions

InvalidOperationException

The current type does not represent a type parameter. That is, IsGenericParameter returns `false`.

Examples

The following example defines a generic class with two type parameters and defines a second generic class that derives from the first class. The derived class's base class has two type arguments: the first is Int32, and the second is a type parameter of the derived type. The example displays information about these generic classes, including the positions reported by the GenericParameterPosition property.

```
C#

using System;
using System.Reflection;
```

```
using System.Collections.Generic;

// Define a base class with two type parameters.
public class Base<T, U> { }

// Define a derived class. The derived class inherits from a constructed
// class that meets the following criteria:
//   (1) Its generic type definition is Base<T, U>.
//   (2) It specifies int for the first type parameter.
//   (3) For the second type parameter, it uses the same type that is used
//       for the type parameter of the derived class.
// Thus, the derived class is a generic type with one type parameter, but
// its base class is an open constructed type with one type argument and
// one type parameter.
public class Derived<V> : Base<int, V> { }

public class Test
{
    public static void Main()
    {
        Console.WriteLine(
            "\r\n--- Display a generic type and the open constructed");
        Console.WriteLine("    type from which it is derived.");

        // Create a Type object representing the generic type definition
        // for the Derived type, by omitting the type argument. (For
        // types with multiple type parameters, supply the commas but
        // omit the type arguments.)
        //
        Type derivedType = typeof(Derived<>);
        DisplayGenericTypeInfo(derivedType);

        // Display its open constructed base type.
        DisplayGenericTypeInfo(derivedType.BaseType);
    }

    private static void DisplayGenericTypeInfo(Type t)
    {
        Console.WriteLine("\r\n{0}", t);

        Console.WriteLine("\tIs this a generic type definition? {0}",
            t.IsGenericTypeDefinition);

        Console.WriteLine("\tIs it a generic type? {0}",
            t.IsGenericType);

        Console.WriteLine("\tDoes it have unassigned generic parameters?
{0}",
            t.ContainsGenericParameters);

        if (t.IsGenericType)
        {
            // If this is a generic type, display the type arguments.
            //
            Type[] typeArguments = t.GetGenericArguments();
```

```
            Console.WriteLine("\tList type arguments ({0}):",
                typeArguments.Length);

            foreach (Type tParam in typeArguments)
            {
                // IsGenericParameter is true only for generic type
                // parameters.
                //
                if (tParam.IsGenericParameter)
                {
                    Console.WriteLine(
                        "\t\t{0}  (unassigned - parameter position {1})",
                        tParam,
                        tParam.GenericParameterPosition);
                }
                else
                {
                    Console.WriteLine("\t\t{0}", tParam);
                }
            }
        }
    }
}

/* This example produces the following output:

--- Display a generic type and the open constructed
    type from which it is derived.

Derived`1[V]
        Is this a generic type definition? True
        Is it a generic type? True
        Does it have unassigned generic parameters? True
        List type arguments (1):
                V  (unassigned - parameter position 0)

Base`2[System.Int32,V]
        Is this a generic type definition? False
        Is it a generic type? True
        Does it have unassigned generic parameters? True
        List type arguments (2):
                System.Int32
                V  (unassigned - parameter position 0)
 */
```

Remarks

The GenericParameterPosition property returns the position of a type parameter in the parameter list of the generic type definition or generic method definition where the type

parameter was originally defined. The DeclaringType and DeclaringMethod properties identify the generic type or method definition:

- If the DeclaringMethod property returns a MethodInfo, that MethodInfo represents a generic method definition, and the current Type object represents a type parameter of that generic method definition.
- If the DeclaringMethod property returns `null`, then the DeclaringType property always returns a Type object representing a generic type definition, and the current Type object represents a type parameter of that generic type definition.

To provide the correct context for the value of the GenericParameterPosition property, it is necessary to identify the generic type or method a type parameter belongs to. For example, consider the return value of the generic method `GetSomething` in the following code:

C#

```
public class B<T, U> { }
public class A<V>
{
    public B<V, X> GetSomething<X>()
    {
        return new B<V, X>();
    }
}
```

The type returned by `GetSomething` depends on the type arguments supplied to class `A` and to `GetSomething` itself. You can obtain a MethodInfo for `GetSomething`, and from that you can obtain the return type. When you examine the type parameters of the return type, GenericParameterPosition returns 0 for both. The position of `V` is 0 because `V` is the first type parameter in the type parameter list for class `A`. The position of `X` is 0 because `X` is the first type parameter in the type parameter list for `GetSomething`.

> **Note**
>
> Calling the **GenericParameterPosition** property causes an exception if the current **Type** does not represent a type parameter. When you examine the type arguments of an open constructed type, use the **IsGenericParameter** property to tell which are type parameters and which are types. The **IsGenericParameter** property returns `true` for a type parameter; you can then use the **GenericParameterPosition** method to obtain its position and use the **DeclaringMethod** and **DeclaringType** properties to determine the generic method or type definition that defines it.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MakeGenericType(Type[])
- ContainsGenericParameters
- IsGenericParameter
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GenericTypeArguments Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an array of the generic type arguments for this type.

C#

```
public virtual Type[] GenericTypeArguments { get; }
```

Property Value

Type[]

An array of the generic type arguments for this type.

Remarks

This property gets only the generic type arguments; that is, the types that have been specified for the generic type parameters of the current type. If the current type is a generic type definition, this property returns an empty array.

> **Note**
>
> If a generic type is used in a generic method or in another generic type, some of its generic type arguments might be generic type parameters of the enclosing method or type.

To get the generic type parameters of a type that represents a generic type definition, use the TypeInfo.GenericTypeParameters property. To get a TypeInfo object for the current Type object, use the IntrospectionExtensions.GetTypeInfo extension method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GUID Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the GUID associated with the Type.

C#

```
public abstract Guid GUID { get; }
```

Property Value

Guid

The GUID associated with the Type.

Examples

The following example creates the class `MyClass1` with a public method, creates a `Type` object corresponding to `MyClass1`, and gets the Guid structure using the `GUID` property of the `Type` class.

C#

```
using System;

class MyGetTypeFromCLSID
{
    public class MyClass1
    {
        public void MyMethod1()
        {
        }
    }
    public static void Main()
    {
        // Get the type corresponding to the class MyClass.
        Type myType = typeof(MyClass1);
        // Get the object of the Guid.
        Guid myGuid =(Guid) myType.GUID;
```

```
        Console.WriteLine("The name of the class is "+myType.ToString());
        Console.WriteLine("The ClassId of MyClass is "+myType.GUID);
    }
}
```

Remarks

A GUID is associated with a type using the GuidAttribute attribute.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Guid

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.HasElementType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.

C#

```
public bool HasElementType { get; }
```

Property Value

Boolean

`true` if the Type is an array, a pointer, or is passed by reference; otherwise, `false`.

Examples

The following example returns `true` or `false` depending on whether or not the object is an array, a reference type, or a pointer.

C#

```
// This code must be compiled with the /unsafe switch:
//   csc /unsafe source.cs
using System;
using System.Reflection;

public class Example
{
    // This method is for demonstration purposes.
    unsafe public void Test(ref int x, out int y, int* z)
    {
        *z = x = y = 0;
    }

    public static void Main()
    {
        // All of the following display 'True'.
```

```
        // Define an array, get its type, and display HasElementType.
        int[] nums = {1, 1, 2, 3, 5, 8, 13};
        Type t = nums.GetType();
        Console.WriteLine("HasElementType is '{0}' for array types.",
t.HasElementType);

        // Test an array type without defining an array.
        t = typeof(Example[]);
        Console.WriteLine("HasElementType is '{0}' for array types.",
t.HasElementType);

        // When you use Reflection Emit to emit dynamic methods and
        // assemblies, you can create array types using MakeArrayType.
        // The following creates the type 'array of Example'.
        t = typeof(Example).MakeArrayType();
        Console.WriteLine("HasElementType is '{0}' for array types.",
t.HasElementType);

        // When you reflect over methods, HasElementType is true for
        // ref, out, and pointer parameter types. The following
        // gets the Test method, defined above, and examines its
        // parameters.
        MethodInfo mi = typeof(Example).GetMethod("Test");
        ParameterInfo[] parms = mi.GetParameters();
        t = parms[0].ParameterType;
        Console.WriteLine("HasElementType is '{0}' for ref parameter
types.", t.HasElementType);
        t = parms[1].ParameterType;
        Console.WriteLine("HasElementType is '{0}' for out parameter
types.", t.HasElementType);
        t = parms[2].ParameterType;
        Console.WriteLine("HasElementType is '{0}' for pointer parameter
types.", t.HasElementType);

        // When you use Reflection Emit to emit dynamic methods and
        // assemblies, you can create pointer and ByRef types to use
        // when you define method parameters.
        t = typeof(Example).MakePointerType();
        Console.WriteLine("HasElementType is '{0}' for pointer types.",
t.HasElementType);
        t = typeof(Example).MakeByRefType();
        Console.WriteLine("HasElementType is '{0}' for ByRef types.",
t.HasElementType);
    }
}
```

Remarks

For example, Type.GetType("Int32[]").HasElementType returns `true`, but Type.GetType("Int32").HasElementType returns `false`. HasElementType also returns

`true` for "Int32*" and "Int32&".

If the current Type represents a generic type, or a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- HasElementTypeImpl()
- IsArray
- IsPointer
- IsByRef
- GetElementType()
- GetType()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsAbstract Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is abstract and must be overridden.

```
C#
public bool IsAbstract { get; }
```

Property Value

Boolean

`true` if the Type is abstract; otherwise, `false`.

Examples

The following example creates an array of Type objects that represent the following types:contains type returns `true` if the specified object is `abstract`; otherwise, it returns `false`.

- `AbstractClass`, an abstract class (a class marked as `abstract` in C# and `MustInherit` in Visual Basic).
- `DerivedClass`, a class that inherits from `AbstractClass`.
- `SingleClass`, a non-inheritable class. It is defined as `sealed` in C# and `NotInheritable` in Visual Basic.
- `ITypeInfo`, an interface.
- `ImplementingClass`, a class that implements the `ITypeInfo` interface.

The method returns `true` only for `AbstractClass`, the abstract class, and `ITypeInfo`, the interface.

C#

```
using System;

public abstract class AbstractClass
{}

public class DerivedClass : AbstractClass
{}

public sealed class SingleClass
{}

public interface ITypeInfo
{
   string GetName();
}

public class ImplementingClass : ITypeInfo
{
   public string GetName()
   {
      return this.GetType().FullName;
   }
}

delegate string InputOutput(string inp);

public class Example
{
   public static void Main()
   {
      Type[] types= { typeof(AbstractClass),
                      typeof(DerivedClass),
                      typeof(ITypeInfo),
                      typeof(SingleClass),
                      typeof(ImplementingClass),
                      typeof(InputOutput) };
      foreach (var type in types)
         Console.WriteLine("{0} is abstract: {1}",
                           type.Name, type.IsAbstract);
   }
}
// The example displays the following output:
//       AbstractClass is abstract: True
//       DerivedClass is abstract: False
//       ITypeInfo is abstract: True
//       SingleClass is abstract: False
//       ImplementingClass is abstract: False
//       InputOutput is abstract: False
```

Remarks

The IsAbstract property returns `true` in the following cases:

- The current type is abstract; that is, it cannot be instantiated, but can only serve as the base class for derived classes. In C#, abstract classes are marked with the abstract keyword; in F#, they are marked with the AbstractClass attribute; in Visual Basic, they are marked with the MustInherit keyword.
- The current type is an interface.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsAnsiClass Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the string format attribute `AnsiClass` is selected for the Type.

C#

```
public bool IsAnsiClass { get; }
```

Property Value

Boolean

`true` if the string format attribute `AnsiClass` is selected for the Type; otherwise, `false`.

Examples

The following example gets the field information and checks for the `AnsiClass` attribute.

C#

```
using System;
using System.Reflection;
public class MyClass
{
    protected string myField = "A sample protected field." ;
}
public class MyType_IsAnsiClass
{
    public static void Main()
    {
        try
        {
            MyClass myObject = new MyClass();
            // Get the type of the 'MyClass'.
            Type myType = typeof(MyClass);
            // Get the field information and the attributes associated with
MyClass.
```

```
            FieldInfo myFieldInfo = myType.GetField("myField",
BindingFlags.NonPublic|BindingFlags.Instance);
            Console.WriteLine( "\nChecking for the AnsiClass attribute for a
field.\n");
            // Get and display the name, field, and the AnsiClass attribute.
            Console.WriteLine("Name of Class: {0} \nValue of Field: {1}
\nIsAnsiClass = {2}", myType.FullName, myFieldInfo.GetValue(myObject),
myType.IsAnsiClass);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: {0}",e.Message);
        }
    }
}
```

Remarks

The StringFormatMask selects the string format attributes. The string format attributes enhance interoperability by defining how strings should be interpreted.

If the current Type represents a generic type, this property pertains to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter of a generic type, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsUnicodeClass
- IsAutoClass

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsArray Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is an array.

C#

```
public bool IsArray { get; }
```

Property Value

Boolean

`true` if the current type is an array; otherwise, `false`.

Examples

The following example demonstrates using the IsArray property.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;

public class Example
{
   public static void Main()
   {
      Type[] types = { typeof(String), typeof(int[]),
                       typeof(ArrayList), typeof(Array),
                       typeof(List<String>),
                       typeof(IEnumerable<Char>) };
      foreach (var t in types)
         Console.WriteLine("{0,-15} IsArray = {1}", t.Name + ":",
                           t.IsArray);
   }
}
// The example displays the following output:
//       String:         IsArray = False
```

```
//       Int32[]:        IsArray = True
//       ArrayList:      IsArray = False
//       Array:          IsArray = False
//       List`1:         IsArray = False
//       IEnumerable`1:  IsArray = False
```

Remarks

The IsArray property returns `false` for the Array class. It also returns `false` if the current instance is a Type object that represents a collection type or an interface designed to work with collections, such as IEnumerable or IEnumerable<T>.

To check for an array, use code such as:

C#

```
typeof(Array).IsAssignableFrom(type)
```

If the current type represents a generic type, or a type parameter in the definition of a generic type or generic method, this property always returns `false`.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsArrayImpl()

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsAutoClass Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the string format attribute `AutoClass` is selected for the Type.

```
C#
public bool IsAutoClass { get; }
```

Property Value

Boolean

`true` if the string format attribute `AutoClass` is selected for the Type; otherwise, `false`.

Remarks

The StringFormatMask selects the string format attributes. The string format attributes enhance interoperability by defining how strings should be interpreted.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter of a generic type, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsAnsiClass
- IsUnicodeClass

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsAutoLayout Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the fields of the current type are laid out automatically by the common language runtime.

C#

```
public bool IsAutoLayout { get; }
```

Property Value

Boolean

`true` if the Attributes property of the current type includes AutoLayout; otherwise, `false`.

Examples

The following example creates an instance of the type and displays the IsAutoLayout property.

C#

```
using System;
using System.Runtime.InteropServices;

// The Demo class is attributed as AutoLayout.
[StructLayoutAttribute(LayoutKind.Auto)]
public class Demo
{
}

public class Example
{
    public static void Main()
    {
        // Create an instance of the Type class using the GetType method.
        Type  myType=typeof(Demo);
```

```
        // Get and display the IsAutoLayout property of the
        // Demoinstance.
        Console.WriteLine("\nThe AutoLayout property for the Demo class is
{0}.",
            myType.IsAutoLayout);
    }
}
```

Remarks

This property is provided as a convenience. Alternatively, you can use the TypeAttributes.LayoutMask enumeration value to select the type layout attributes, and then test whether TypeAttributes.AutoLayout is set. The TypeAttributes.AutoLayout,TypeAttributes.ExplicitLayout, and TypeAttributes.SequentialLayout enumeration values indicate the way the fields of the type are laid out in memory.

For dynamic types, you can specify TypeAttributes.AutoLayout when you create the type. In code, apply the StructLayoutAttribute attribute with the LayoutKind.Auto enumeration value to the type, to let the runtime determine the appropriate way to lay out the class.

> ⓘ **Note**
>
> You cannot use the **GetCustomAttributes** method to determine whether the **StructLayoutAttribute** has been applied to a type.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>.`

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsLayoutSequential
- IsExplicitLayout
- Metadata and Self-Describing Components

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsByRef Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is passed by reference.

C#

```
public bool IsByRef { get; }
```

Property Value

Boolean

`true` if the Type is passed by reference; otherwise, `false`.

Examples

The following example demonstrates a use of the `IsByRef` property to check whether a specified type is passed by reference. The example defines the class `MyTypeDelegator`, which overrides the `HasElementTypeImpl` method. The main class checks for the `HasElementType` property and displays the element type.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegator : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;
    public MyTypeDelegator(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override Type.HasElementTypeImpl().
    protected override bool HasElementTypeImpl()
    {
        // Determine whether the type is an array.
```

```
        if(myType.IsArray)
        {
            myElementType = "array";
            return true;
        }
        // Determine whether the type is a reference.
        if(myType.IsByRef)
        {
            myElementType = "reference";
            return true;
        }
        // Determine whether the type is a pointer.
        if(myType.IsPointer)
        {
            myElementType = "pointer";
            return true;
        }
        // Return false if the type is not a reference, array, or pointer
type.
        return false;
    }
}
public class Type_HasElementTypeImpl
{
    public static void Main()
    {
        try
        {
            int myInt = 0 ;
            int[] myArray = new int[5];
            MyTypeDelegator myType = new MyTypeDelegator(myArray.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            Console.WriteLine("\nDetermine whether a variable is an array,
pointer, or reference type.\n");
            if( myType.HasElementType)
                Console.WriteLine("The type of myArray is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myArray is not an array, pointer, or
reference type.");
            myType = new MyTypeDelegator(myInt.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            if( myType.HasElementType)
                Console.WriteLine("The type of myInt is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myInt is not an array, pointer, or
reference type.");
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
        }
```

```
    }
}
```

Remarks

To get to the actual type, dereference the type that was passed by reference, and then call GetElementType on that type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsByRefImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsByRefLike Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is a byref-like structure.

C#

```
public virtual bool IsByRefLike { get; }
```

Property Value

Boolean

`true` if the Type is a byref-like structure; otherwise, `false`.

Remarks

Byref-like structures are declared using `ref struct` keyword in C#. An instance of the byref-like structure can't be placed on the managed heap. For more information, see Ref struct.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.IsClass Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is a class or a delegate; that is, not a value type or interface.

C#

```
public bool IsClass { get; }
```

Property Value

Boolean

`true` if the Type is a class; otherwise, `false`.

Examples

The following example creates an instance of a type and indicates whether the type is a class.

C#

```
using System;
using System.Reflection;

public  class MyDemoClass
{
}

public class MyTypeClass
{
    public static void Main(string[] args)
    {
        try
        {
            Type  myType = typeof(MyDemoClass);
            // Get and display the 'IsClass' property of the 'MyDemoClass'
instance.
```

```
            Console.WriteLine("\nIs the specified type a class? {0}.",
myType.IsClass);
        }
        catch(Exception e)
        {
            Console.WriteLine("\nAn exception occurred: {0}." ,e.Message);
        }
    }
}
```

Remarks

This property returns `true` for classes as well as delegates. It returns `false` for value types (for structures and enumerations) even if they are boxed.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `true`. If the current Type represents a constructed generic type, this property returns `true` if the generic type definition is a class definition; that is, it does not define an interface or a value type.

> **Note**
>
> This property returns `true` for `Type` instances that represent the **Enum** and **ValueType** classes. These two classes are the base types for enumerations and value types, respectively, but they are not enumerations or value types themselves. For more information, see the **IsValueType** and **IsEnum** properties.

The TypeAttributes.ClassSemanticsMask enumeration value distinguishes a type declaration as class or interface. However, both classes and value types are marked with the TypeAttributes.Class attribute. If you retrieve the value of a type's Attributes property and use the TypeAttributes.ClassSemanticsMask value to determine whether a type is a class instead of a value type, you must also call the IsValueType property. The example for the TypeAttributes enumeration contains additional information as well as anexample.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsInterface
- IsValueType
- ClassSemanticsMask

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsCOMObject Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is a COM object.

C#

```
public bool IsCOMObject { get; }
```

Property Value

Boolean

`true` if the Type is a COM object; otherwise, `false`.

Remarks

This method returns `false` for COM interfaces because they are not objects. COM interfaces can be implemented by Microsoft .NET Framework objects.

You can also load a COM class and get a `Type` object for that COM class by using the Tlbimp.exe (Type Library Importer) tool.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsCOMObjectImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsConstructedGenericType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether this object represents a constructed generic type. You can create instances of a constructed generic type.

```
C#
public virtual bool IsConstructedGenericType { get; }
```

Property Value

Boolean

`true` if this object represents a constructed generic type; otherwise, `false`.

Remarks

A constructed generic type has had explicit types supplied for all of its generic type parameters. It is also referred to as a closed generic type.

When this property is `true`, you can create instances of the current type; when it is `false`, you can't.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsContextful Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type can be hosted in a context.

C#

```
public bool IsContextful { get; }
```

Property Value

Boolean

`true` if the Type can be hosted in a context; otherwise, `false`.

Examples

The following example demonstrates the `IsContextful`, IsMarshalByRef, and IsPrimitive properties of the Type class. It checks whether the given type can be hosted in the context, whether it can be marshaled by reference, and whether the type is a primitive data type.

C#

```
using System;
using System.Runtime.Remoting.Contexts;

public class ContextBoundClass: ContextBoundObject
{
    public string Value = "The Value property.";
}

public class Example
{
    public static void Main()
    {
         // Determine whether the types can be hosted in a Context.
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
```

```
                            typeof(Example).Name,
typeof(Example).IsContextful);
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsContextful);

         // Determine whether the types are marshalled by reference.
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(Example).Name,
typeof(Example).IsMarshalByRef);
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsMarshalByRef);

         // Determine whether the types are primitive datatypes.
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(int).Name, typeof(int).IsPrimitive);
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(string).Name, typeof(string).IsPrimitive);
    }
}
// The example displays the following output:
//    The IsContextful property for the Example type is False.
//    The IsContextful property for the ContextBoundClass type is True.
//    The IsMarshalByRef property of Example is False.
//    The IsMarshalByRef property of ContextBoundClass is True.
//    Int32 is a primitive data type: True.
//    String is a primitive data type: False.
```

Remarks

A context intercepts calls to the class members and enforces policies that are applied to the class, such as synchronization. For more detailed information on remoting contexts, see Context.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsContextfulImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsEnum Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type represents an enumeration.

C#

```
public virtual bool IsEnum { get; }
```

Property Value

Boolean

`true` if the current Type represents an enumeration; otherwise, `false`.

Examples

The following example demonstrates how to use the `IsEnum` property.

C#

```
using System;
public enum Color
{ Red, Blue, Green }

class TestIsEnum
{
    public static void Main()
    {
       Type colorType = typeof(Color);
       Type enumType = typeof(Enum);
       Console.WriteLine("Is Color an enum? {0}.", colorType.IsEnum);
       Console.WriteLine("Is Color a value type? {0}.",
colorType.IsValueType);
       Console.WriteLine("Is Enum an enum Type? {0}.", enumType.IsEnum);
       Console.WriteLine("Is Enum a value type? {0}.",
enumType.IsValueType);
   }
}
// The example displays the following output:
```

```
//     Is Color an enum? True.
//     Is Color a value type? True.
//     Is Enum an enum type? False.
//     Is Enum a value type? False.
```

Remarks

This property returns `true` for an enumeration, but not for the Enum type itself.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Type.IsExplicitLayout Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the fields of the current type are laid out at explicitly specified offsets.

C#

```
public bool IsExplicitLayout { get; }
```

Property Value

Boolean

`true` if the Attributes property of the current type includes ExplicitLayout; otherwise, `false`.

Examples

The following example creates an instance of a type and displays the value of its IsExplicitLayout property. It uses the `MySystemTime` class, which is also in the code example for StructLayoutAttribute.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;
using System.Runtime.InteropServices;

// Class to test for the ExplicitLayout property.
[StructLayout(LayoutKind.Explicit, Size=16, CharSet=CharSet.Ansi)]
public class MySystemTime
{
   [FieldOffset(0)]public ushort wYear;
   [FieldOffset(2)]public ushort wMonth;
   [FieldOffset(4)]public ushort wDayOfWeek;
   [FieldOffset(6)]public ushort wDay;
```

```
    [FieldOffset(8)]public ushort wHour;
    [FieldOffset(10)]public ushort wMinute;
    [FieldOffset(12)]public ushort wSecond;
    [FieldOffset(14)]public ushort wMilliseconds;
}

public class Program
{
    public static void Main(string[] args)
    {
        // Create an instance of the type using the GetType method.
        Type  t = typeof(MySystemTime);
        // Get and display the IsExplicitLayout property.
        Console.WriteLine("\nIsExplicitLayout for MySystemTime is {0}.",
            t.IsExplicitLayout);
    }
}
```

Remarks

This property is provided as a convenience. Alternatively, you can use the TypeAttributes.LayoutMask enumeration value to select the type layout attributes, and then test whether TypeAttributes.ExplicitLayout is set. The TypeAttributes.AutoLayout, TypeAttributes.ExplicitLayout, and TypeAttributes.SequentialLayout enumeration values indicate the way the fields of the type are laid out in memory.

For dynamic types, you can specify TypeAttributes.ExplicitLayout when you create the type. In code, apply the StructLayoutAttribute attribute with the LayoutKind.Explicit enumeration value to the type, to specify that the offsets at which the fields start are specified explicitly.

> ⓘ **Note**
>
> You cannot use the **GetCustomAttributes** method to determine whether the **StructLayoutAttribute** has been applied to a type.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- StructLayoutAttribute
- IsAutoLayout
- IsLayoutSequential
- Metadata and Self-Describing Components

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsFunctionPointer Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current Type is a function pointer.

C#

```
public virtual bool IsFunctionPointer { get; }
```

Property Value

Boolean

`true` if the current Type is a function pointer; otherwise, `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsGenericMethodParameter Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic method.

C#

```
public virtual bool IsGenericMethodParameter { get; }
```

Property Value

Boolean

`true` if the Type object represents a type parameter of a generic method definition; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Type.IsGenericParameter Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type represents a type parameter in the definition of a generic type or method.

C#

```
public virtual bool IsGenericParameter { get; }
```

Property Value

Boolean

`true` if the Type object represents a type parameter of a generic type definition or generic method definition; otherwise, `false`.

Examples

The following example uses the IsGenericParameter property to test for generic type parameters in a generic type.

C#

```
if (t.IsGenericType)
{
    // If this is a generic type, display the type arguments.
    //
    Type[] typeArguments = t.GetGenericArguments();

    Console.WriteLine("\tList type arguments ({0}):",
        typeArguments.Length);

    foreach (Type tParam in typeArguments)
    {
        // If this is a type parameter, display its
        // position.
        //
        if (tParam.IsGenericParameter)
```

```
        {
            Console.WriteLine("\t\t{0}\t(unassigned - parameter position
{1})",
                tParam,
                tParam.GenericParameterPosition);
        }
        else
        {
            Console.WriteLine("\t\t{0}", tParam);
        }
    }
}
```

Remarks

Type objects that represent generic type parameters can be obtained by calling the GetGenericArguments method of a Type object that represents a generic type definition, or the GetGenericArguments method of a MethodInfo object that represents a generic method definition.

- For a generic type or method definition, the IsGenericParameter property returns `true` for every element of the resulting array.
- For a closed constructed type or method, the IsGenericParameter property returns `false` for every element of the array returned by the GetGenericArguments method.
- For an open constructed type or method, some elements of the array might be specific types and others might be type parameters. IsGenericParameter returns `false` for the types and `true` for the type parameters. The code example for the ContainsGenericParameters property demonstrates a generic class with a mixture of types and type parameters.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- GenericParameterPosition
- GetGenericArguments()
- GetGenericParameterConstraints()
- GenericParameterAttributes
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsGenericType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current type is a generic type.

C#

```
public virtual bool IsGenericType { get; }
```

Property Value

Boolean

`true` if the current type is a generic type; otherwise, `false`.

Examples

The following code example displays the value of the IsGenericType, IsGenericTypeDefinition, IsGenericParameter, and ContainsGenericParameters properties for the types described in the Remarks section. For explanations of the property values, see the accompanying table in Remarks.

C#

```
using System;
using System.Reflection;

public class Base<T, U> {}

public class Derived<V> : Base<string, V>
{
    public G<Derived <V>> F;

    public class Nested {}
}

public class G<T> {}

class Example
```

```
{
    public static void Main()
    {
        // Get the generic type definition for Derived, and the base
        // type for Derived.
        //
        Type tDerived = typeof(Derived<>);
        Type tDerivedBase = tDerived.BaseType;

        // Declare an array of Derived<int>, and get its type.
        //
        Derived<int>[] d = new Derived<int>[0];
        Type tDerivedArray = d.GetType();

        // Get a generic type parameter, the type of a field, and a
        // type that is nested in Derived. Notice that in order to
        // get the nested type it is necessary to either (1) specify
        // the generic type definition Derived<>, as shown here,
        // or (2) specify a type parameter for Derived.
        //
        Type tT = typeof(Base<,>).GetGenericArguments()[0];
        Type tF = tDerived.GetField("F").FieldType;
        Type tNested = typeof(Derived<>.Nested);

        DisplayGenericType(tDerived, "Derived<V>");
        DisplayGenericType(tDerivedBase, "Base type of Derived<V>");
        DisplayGenericType(tDerivedArray, "Array of Derived<int>");
        DisplayGenericType(tT, "Type parameter T from Base<T>");
        DisplayGenericType(tF, "Field type, G<Derived<V>>");
        DisplayGenericType(tNested, "Nested type in Derived<V>");
    }

    public static void DisplayGenericType(Type t, string caption)
    {
        Console.WriteLine("\n{0}", caption);
        Console.WriteLine("    Type: {0}", t);

        Console.WriteLine("\t            IsGenericType: {0}",
            t.IsGenericType);
        Console.WriteLine("\t  IsGenericTypeDefinition: {0}",
            t.IsGenericTypeDefinition);
        Console.WriteLine("\tContainsGenericParameters: {0}",
            t.ContainsGenericParameters);
        Console.WriteLine("\t       IsGenericParameter: {0}",
            t.IsGenericParameter);
    }
}

/* This code example produces the following output:

Derived<V>
    Type: Derived`1[V]
                    IsGenericType: True
          IsGenericTypeDefinition: True
        ContainsGenericParameters: True
```

```
                IsGenericParameter: False

Base type of Derived<V>
    Type: Base`2[System.String,V]
                    IsGenericType: True
          IsGenericTypeDefinition: False
        ContainsGenericParameters: True
                IsGenericParameter: False

Array of Derived<int>
    Type: Derived`1[System.Int32][]
                    IsGenericType: False
          IsGenericTypeDefinition: False
        ContainsGenericParameters: False
                IsGenericParameter: False

Type parameter T from Base<T>
    Type: T
                    IsGenericType: False
          IsGenericTypeDefinition: False
        ContainsGenericParameters: True
                IsGenericParameter: True

Field type, G<Derived<V>>
    Type: G`1[Derived`1[V]]
                    IsGenericType: True
          IsGenericTypeDefinition: False
        ContainsGenericParameters: True
                IsGenericParameter: False

Nested type in Derived<V>
    Type: Derived`1+Nested[V]
                    IsGenericType: True
          IsGenericTypeDefinition: True
        ContainsGenericParameters: True
                IsGenericParameter: False
 */
```

Remarks

Use the IsGenericType property to determine whether a Type object represents a generic type. Use the ContainsGenericParameters property to determine whether a Type object represents an open constructed type or a closed constructed type.

> ⓘ **Note**

The **IsGenericType** property returns `false` if the immediate type is not generic. For example, an array whose elements are of type `A<int>` (`A(Of Integer)` in Visual Basic) is not itself a generic type.

The following table summarizes the invariant conditions for common terms used in generic reflection.

Expand table

Term	Invariant
generic type definition	The IsGenericTypeDefinition property is `true`. Defines a generic type. A constructed type is created by calling the MakeGenericType method on a Type object that represents a generic type definition and specifying an array of type arguments. MakeGenericType can be called only on generic type definitions. Any generic type definition is a generic type (the IsGenericType property is `true`), but the converse is not true.
generic type	The IsGenericType property is `true`. Can be a generic type definition, an open constructed type, or a closed constructed type. Note that an array type whose element type is generic is not itself a generic type. The same is true of a Type object representing a pointer to a generic type.
open constructed type	The ContainsGenericParameters property is `true`. Examples are a generic type that has unassigned type parameters, a type that is nested in a generic type definition or in an open constructed type, or a generic type that has a type argument for which the ContainsGenericParameters property is `true`. It is not possible to create an instance of an open constructed type. Note that not all open constructed types are generic. For example, an array whose element type is a generic type definition is not generic, and a pointer to an open constructed type is not generic.
closed constructed type	The ContainsGenericParameters property is `false`. When examined recursively, the type has no unassigned generic parameters.

Term	Invariant
generic type parameter	The IsGenericParameter property is `true`. The ContainsGenericParameters property is `true`. In a generic type definition, a placeholder for a type that will be assigned later.
generic type argument	Can be any type, including a generic type parameter. Type arguments are specified as an array of Type objects passed to the MakeGenericType method when creating a constructed generic type. If instances of the resulting type are to be created, the ContainsGenericParameters property must be `false` for all the type arguments.

The following code example and table illustrate some of these terms and invariants. The `Derived` class is of particular interest because its base type is a constructed type that has a mixture of types and type parameters in its type argument list.

C#

```
public class Base<T, U> {}

public class Derived<V> : Base<string, V>
{
    public G<Derived <V>> F;

    public class Nested {}
}

public class G<T> {}
```

The following table shows examples that use and build on the classes `Base`, `Derived`, and `G`. When the C++ and C# code is the same, only one entry is shown.

Expand table

Example	Invariants

`Derived(Of V)`
 	
`Derived<V>`
	For this type:
	
	IsGenericType is `true`.
	
	IsGenericTypeDefinition is `true`.
	
	ContainsGenericParameters is `true`.

Example	Invariants

`Base(Of String, V)`
 	
`Base<String,V>`
 	
`Base<String^,V>`
	For this type:
	
	IsGenericType is `true`.
	
	IsGenericTypeDefinition is `false`.
	
	ContainsGenericParameters is `true`.

`Dim d() As Derived(Of Integer)`
 	
`Derived<int>[] d;`
 	
`array<Derived<int>^>^ d;`
	For the type of variable `d`:
	
	IsGenericType is `false` because `d` is an array.
	
	IsGenericTypeDefinition is `false`.
	
	ContainsGenericParameters is `false`.

`T`, `U`, and `V` (everywhere they appear)	IsGenericParameter is `true`. IsGenericType is `false` because there is no way to constrain a type parameter to generic types. IsGenericTypeDefinition is `false`. ContainsGenericParameters is `true` because `T`, `U`, and `V` are themselves generic type parameters. This does not imply anything about type arguments that are assigned to them later.
The type of field `F`	IsGenericType is `true`. IsGenericTypeDefinition is `false` because a type has been assigned to the type parameter of `G`. Note that this is equivalent to having called the MakeGenericType method. ContainsGenericParameters is `true` because the type of field `F` has a type argument that is an open constructed type. The constructed type is open because its type argument (that is, `Base`) is a generic type definition. This illustrates the recursive nature of the IsGenericType property.
The nested class `Nested`	IsGenericType is `true`, even though the `Nested` class has no generic type parameters of its own, because it is nested in a generic type. IsGenericTypeDefinition is `true`. That is, you can call the MakeGenericType method and supply the type parameter of the enclosing type, `Derived`. ContainsGenericParameters is `true` because the enclosing type,

Example	Invariants
	`Derived`, has generic type parameters. This illustrates the recursive nature of the ContainsGenericParameters property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ContainsGenericParameters
- IsGenericTypeDefinition
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsGenericTypeDefinition Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type represents a generic type definition, from which other generic types can be constructed.

C#

```
public virtual bool IsGenericTypeDefinition { get; }
```

Property Value

Boolean

`true` if the Type object represents a generic type definition; otherwise, `false`.

Examples

The following example displays information about a type, including whether or not it is a generic type definition. Information is displayed for a constructed type, for its generic type definition, and for an ordinary type.

C#

```
using System;
using System.Reflection;
using System.Collections.Generic;

public class Test
{
    private static void DisplayGenericTypeInfo(Type t)
    {
        Console.WriteLine("\r\n{0}", t);

        Console.WriteLine("\tIs this a generic type definition? {0}",
            t.IsGenericTypeDefinition);

        Console.WriteLine("\tIs it a generic type? {0}",
            t.IsGenericType);
```

```
        if (t.IsGenericType)
        {
            // If this is a generic type, display the type arguments.
            //
            Type[] typeArguments = t.GetGenericArguments();

            Console.WriteLine("\tList type arguments ({0}):",
                typeArguments.Length);

            foreach (Type tParam in typeArguments)
            {
                // If this is a type parameter, display its
                // position.
                //
                if (tParam.IsGenericParameter)
                {
                    Console.WriteLine("\t\t{0}\t(unassigned - parameter
position {1})",
                        tParam,
                        tParam.GenericParameterPosition);
                }
                else
                {
                    Console.WriteLine("\t\t{0}", tParam);
                }
            }
        }
    }

    public static void Main()
    {
        Console.WriteLine("\r\n--- Display information about a constructed
type, its");
        Console.WriteLine("    generic type definition, and an ordinary
type.");

        // Create a Dictionary of Test objects, using strings for the
        // keys.
        Dictionary<string, Test> d = new Dictionary<string, Test>();

        // Display information for the constructed type and its generic
        // type definition.
        DisplayGenericTypeInfo(d.GetType());
        DisplayGenericTypeInfo(d.GetType().GetGenericTypeDefinition());

        // Display information for an ordinary type.
        DisplayGenericTypeInfo(typeof(string));
    }
}

/* This example produces the following output:

--- Display information about a constructed type, its
    generic type definition, and an ordinary type.
```

```
System.Collections.Generic.Dictionary[System.String,Test]
        Is this a generic type definition? False
        Is it a generic type? True
        List type arguments (2):
                System.String
                Test

System.Collections.Generic.Dictionary[TKey,TValue]
        Is this a generic type definition? True
        Is it a generic type? True
        List type arguments (2):
                TKey    (unassigned - parameter position 0)
                TValue  (unassigned - parameter position 1)

System.String
        Is this a generic type definition? False
        Is it a generic type? False
 */
```

Remarks

A generic type definition is a template from which other types can be constructed. For example, from the generic type definition `G<T>` (expressed in C# syntax; `G(Of T)` in Visual Basic or `generic <typename T> ref class G` in C++) you can construct and instantiate the type `G<int>` (`G(Of Integer)` in Visual Basic), by calling the MakeGenericType method with a generic argument list containing the Int32 type. Given a Type object representing this constructed type, the GetGenericTypeDefinition method gets the generic type definition back again.

Use the IsGenericTypeDefinition property to determine whether you can create new types from the current type. If the IsGenericTypeDefinition property returns `true`, you can call the MakeGenericType method to create new generic types.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ContainsGenericParameters
- GetGenericTypeDefinition()
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsGenericTypeParameter Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic type.

C#

```
public virtual bool IsGenericTypeParameter { get; }
```

Property Value

Boolean

`true` if the Type object represents a type parameter of a generic type definition; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsImport Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type has a ComImportAttribute attribute applied, indicating that it was imported from a COM type library.

```
C#
public bool IsImport { get; }
```

Property Value

Boolean

`true` if the Type has a ComImportAttribute; otherwise, `false`.

Remarks

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>.`

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsInterface Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is an interface; that is, not a class or a value type.

C#

```
public bool IsInterface { get; }
```

Property Value

Boolean

`true` if the Type is an interface; otherwise, `false`.

Examples

The following example creates an interface, checks for the interface type, and indicates whether a class has the `IsInterface` property set.

C#

```
using System;
// Declare an interface.
interface myIFace
{
}
class MyIsInterface
{
    public static void Main(string []args)
    {
        try
        {
            // Get the IsInterface attribute for myIFace.
            bool myBool1 = typeof(myIFace).IsInterface;
            //Display the IsInterface attribute for myIFace.
            Console.WriteLine("Is the specified type an interface? {0}.",
myBool1);
```

```
            // Get the attribute IsInterface for MyIsInterface.
            bool myBool2 = typeof(MyIsInterface).IsInterface;
            //Display the IsInterface attribute for MyIsInterface.
            Console.WriteLine("Is the specified type an interface? {0}.",
myBool2);
        }
        catch(Exception e)
        {
            Console.WriteLine("\nAn exception occurred: {0}.", e.Message);
        }
    }
}
/* The example produces the following output:

Is the specified type an interface? True.
Is the specified type an interface? False.
*/
```

Remarks

The ClassSemanticsMask distinguishes a type declaration as class, interface or value type.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsClass
- IsValueType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsLayoutSequential Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the fields of the current type are laid out sequentially, in the order that they were defined or emitted to the metadata.

C#

```
public bool IsLayoutSequential { get; }
```

Property Value

Boolean

`true` if the Attributes property of the current type includes SequentialLayout; otherwise, `false`.

Examples

The following example creates an instance of a class for which the LayoutKind.Sequential enumeration value in the StructLayoutAttribute class has been set, checks for the IsLayoutSequential property, and displays the result.

C#

```
using System;
using System.Reflection;
using System.ComponentModel;
using System.Runtime.InteropServices;
class MyTypeSequential1
{
}
[StructLayoutAttribute(LayoutKind.Sequential)]
class MyTypeSequential2
{
    public static void Main(string []args)
    {
        try
```

```
        {
            // Create an instance of myTypeSeq1.
            MyTypeSequential1 myObj1 = new MyTypeSequential1();
            Type myTypeObj1 = myObj1.GetType();
            // Check for and display the SequentialLayout attribute.
            Console.WriteLine("\nThe object myObj1 has IsLayoutSequential:
{0}.", myObj1.GetType().IsLayoutSequential);
            // Create an instance of 'myTypeSeq2' class.
            MyTypeSequential2 myObj2 = new MyTypeSequential2();
            Type myTypeObj2 = myObj2.GetType();
            // Check for and display the SequentialLayout attribute.
            Console.WriteLine("\nThe object myObj2 has IsLayoutSequential:
{0}.", myObj2.GetType().IsLayoutSequential);
        }
        catch(Exception e)
        {
            Console.WriteLine("\nAn exception occurred: {0}", e.Message);
        }
    }
}
```

Remarks

This property is provided as a convenience. Alternatively, you can use the TypeAttributes.LayoutMask enumeration value to select the type layout attributes, and then test whether TypeAttributes.SequentialLayout is set. The TypeAttributes.AutoLayout, TypeAttributes.ExplicitLayout, and TypeAttributes.SequentialLayout enumeration values indicate the way the fields of the type are laid out in memory.

For dynamic types, you can specify TypeAttributes.SequentialLayout when you create the type. In code, apply the StructLayoutAttribute attribute with the LayoutKind.Sequential enumeration value to the type, to specify that layout is sequential.

> ⓘ **Note**
>
> You cannot use the **GetCustomAttributes** method to determine whether the **StructLayoutAttribute** has been applied to a type.

For more information, see section 9.1.2 of the specification for the Common Language Infrastructure (CLI) documentation, "Partition II: Metadata Definition and Semantics".

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current

Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsAutoLayout
- IsExplicitLayout
- Metadata and Self-Describing Components

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsMarshalByRef Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is marshaled by reference.

C#

```
public bool IsMarshalByRef { get; }
```

Property Value

Boolean

`true` if the Type is marshaled by reference; otherwise, `false`.

Examples

The following example demonstrates the IsContextful, `IsMarshalByRef`, and IsPrimitive properties of the Type class. It checks whether the given type can be hosted in the context, whether it can be marshaled by reference, and whether the type is a primitive data type.

C#

```
using System;
using System.Runtime.Remoting.Contexts;

public class ContextBoundClass: ContextBoundObject
{
    public string Value = "The Value property.";
}

public class Example
{
    public static void Main()
    {
         // Determine whether the types can be hosted in a Context.
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
```

```
                            typeof(Example).Name,
typeof(Example).IsContextful);
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsContextful);

         // Determine whether the types are marshalled by reference.
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(Example).Name,
typeof(Example).IsMarshalByRef);
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsMarshalByRef);

         // Determine whether the types are primitive datatypes.
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(int).Name, typeof(int).IsPrimitive);
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(string).Name, typeof(string).IsPrimitive);
    }
}
// The example displays the following output:
//    The IsContextful property for the Example type is False.
//    The IsContextful property for the ContextBoundClass type is True.
//    The IsMarshalByRef property of Example is False.
//    The IsMarshalByRef property of ContextBoundClass is True.
//    Int32 is a primitive data type: True.
//    String is a primitive data type: False.
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsMarshalByRefImpl()

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsNested Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the current Type object represents a type whose definition is nested inside the definition of another type.

C#

```
public bool IsNested { get; }
```

Property Value

Boolean

`true` if the Type is nested inside another type; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}

    protected internal class ProtectedInternalClass
```

```
    {}

    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
```

```
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

The IsNested property returns `true` for all nested types, regardless of visibility. To test for nesting and visibility at the same time, use the related properties IsNestedAssembly,

IsNestedFamily, IsNestedFamANDAssem, IsNestedFamORAssem, IsNestedPrivate, or IsNestedPublic.

> **Note**
>
> The **VisibilityMask** enumeration member selects the visibility attributes for a type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsNestedAssembly
- IsNestedFamily
- IsNestedFamANDAssem
- IsNestedFamORAssem
- IsNestedPrivate
- IsNestedPublic

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedAssembly Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is nested and visible only within its own assembly.

C#

```
public bool IsNestedAssembly { get; }
```

Property Value

Boolean

`true` if the Type is nested and visible only within its own assembly; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}

    protected internal class ProtectedInternalClass
```

```
    {}

    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
```

```
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedFamANDAssem Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is nested and visible only to classes that belong to both its own family and its own assembly.

```
C#

public bool IsNestedFamANDAssem { get; }
```

Property Value

Boolean

`true` if the Type is nested and visible only to classes that belong to both its own family and its own assembly; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

```
C#

using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}
```

```
    protected internal class ProtectedInternalClass
    {}

    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
```

```
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

> **Note**
>
> The C# and Visual Basic languages do not include semantics that allow you to define a nested type that is visible only to protected types in its own assembly. `protected internal` visibility in C# and `Protected Friend` visibility in Visual Basic define a nested type that is visible both to protected types and to types in the same assembly.

A Type object's family is defined as all objects of the same Type and of its subtypes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedFamily Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is nested and visible only within its own family.

C#

```
public bool IsNestedFamily { get; }
```

Property Value

Boolean

`true` if the Type is nested and visible only within its own family; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}

    protected internal class ProtectedInternalClass
    {}
```

```
    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
```

```
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

A Type object's family is defined as all objects of the exact same Type and of its subtypes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedFamORAssem Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is nested and visible only to classes that belong to either its own family or to its own assembly.

C#

```
public bool IsNestedFamORAssem { get; }
```

Property Value

Boolean

`true` if the Type is nested and visible only to classes that belong to its own family or to its own assembly; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}
```

```
    protected internal class ProtectedInternalClass
    {}

    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
```

```
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the visibility of a type is `protected internal` in C# or `Protected Friend` in Visual Basic, the IsNestedFamORAssem property returns `true`.

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

A Type object's family is defined as all objects of the exact same Type and of its subtypes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedPrivate Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is nested and declared private.

C#

```
public bool IsNestedPrivate { get; }
```

Property Value

Boolean

`true` if the Type is nested and declared private; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}

    protected internal class ProtectedInternalClass
    {}
```

```
    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
```

```
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNestedPublic Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether a class is nested and declared public.

C#

```
public bool IsNestedPublic { get; }
```

Property Value

Boolean

`true` if the class is nested and declared public; otherwise, `false`.

Examples

The following example creates an outer class with a number of nested classes that have various types of visibility. It then retrieves the value of a number of visibility-related Type properties for the parent type and each of its nested types.

C#

```
using System;

// Create a class with a number of nested classes.
public class OuterClass
{
    private class PrivateClass
    {}

    protected class ProtectedClass
    {}

    internal class InternalClass
    {}

    protected internal class ProtectedInternalClass
    {}
```

```
    public class PublicClass
    {}

    public static void Main()
    {
        // Create an array of Type objects for all the classes.
        Type[] types = { typeof(OuterClass),
                         typeof(OuterClass.PublicClass),
                         typeof(OuterClass.PrivateClass),
                         typeof(OuterClass.ProtectedClass),
                         typeof(OuterClass.InternalClass),
                         typeof(OuterClass.ProtectedInternalClass) };
        // Display the property values of each nested class.
        foreach (var type in types) {
           Console.WriteLine("\n{0} property values:", type.Name);
           Console.WriteLine("   Public Class: {0}", type.IsPublic);
           Console.WriteLine("   Not a Public Class: {0}",
type.IsNotPublic);
           Console.WriteLine("   Nested Class: {0}", type.IsNested);
           Console.WriteLine("   Nested Private Class: {0}",
type.IsNestedPrivate);
           Console.WriteLine("   Nested Internal Class: {0}",
type.IsNestedAssembly);
           Console.WriteLine("   Nested Protected Class: {0}",
type.IsNestedFamily);
           Console.WriteLine("   Nested Family Or Assembly Class: {0}",
type.IsNestedFamORAssem);
           Console.WriteLine("   Nested Family And Assembly Class: {0}",
type.IsNestedFamANDAssem);
           Console.WriteLine("   Nested Public Class: {0}",
type.IsNestedPublic);
        }
    }
}
// The example displays the following output:
//    OuterClass property values:
//       Public Class: True
//       Not a Public Class: False
//       Nested Class: False
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    PublicClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
```

```
//       Nested Family And Assembly Class: False
//       Nested Public Class: True
//
//    PrivateClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: True
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: True
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    InternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: True
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: False
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
//
//    ProtectedInternalClass property values:
//       Public Class: False
//       Not a Public Class: False
//       Nested Class: True
//       Nested Private Class: False
//       Nested Internal Class: False
//       Nested Protected Class: False
//       Nested Family Or Assembly Class: True
//       Nested Family And Assembly Class: False
//       Nested Public Class: False
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `false`.

TypeAttributes.VisibilityMask selects the visibility attributes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsNotPublic Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is not declared public.

C#

```
public bool IsNotPublic { get; }
```

Property Value

Boolean

`true` if the Type is not declared public and is not a nested type; otherwise, `false`.

Examples

This example uses the `IsNotPublic` property to get the visibility of the type.

C#

```
using System;
using System.IO;
using System.Reflection;

class Example
{
    public static void Main()
    {
        // Get the Type and MemberInfo.
        Type t = Type.GetType("System.IO.File");
        MemberInfo[] members = t.GetMembers();
        // Get and display the DeclaringType method.
        Console.WriteLine("\nThere are {0} members in {1}.",
                          members.Length, t.FullName);
        Console.WriteLine("Is {0} non-public? {1}",
                          t.FullName, t.IsNotPublic);
    }
}
// The example displays output like the following:
```

```
//       There are 60 members in System.IO.File.
//       Is System.IO.File non-public? False
```

The following code example demonstrates why you cannot use `IsPublic` and `IsNotPublic` for nested classes.

```C#
public class A
{
    public class B { }
    private class C { }
}
```

For nested classes, ignore the results of `IsPublic` and `IsNotPublic` and pay attention only to the results of `IsNestedPublic` and `IsNestedPrivate`. The reflection output for this code fragment would be as follows:

Expand table

Class	IsNotPublic	IsPublic	IsNestedPublic	IsNestedPrivate
A	FALSE	TRUE	FALSE	FALSE
B	FALSE	FALSE	TRUE	FALSE
C	FALSE	FALSE	FALSE	TRUE

Remarks

Do not use this property with nested types; use the IsNestedPublic property instead.

If the current Type represents a type parameter of a generic type, this property returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsPointer Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is a pointer.

C#

```
public bool IsPointer { get; }
```

Property Value

Boolean

`true` if the Type is a pointer; otherwise, `false`.

Examples

The following example shows a use of the `IsPointer` property.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegator : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;
    public MyTypeDelegator(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override Type.HasElementTypeImpl().
    protected override bool HasElementTypeImpl()
    {
        // Determine whether the type is an array.
        if(myType.IsArray)
        {
            myElementType = "array";
            return true;
        }
```

```
        // Determine whether the type is a reference.
        if(myType.IsByRef)
        {
            myElementType = "reference";
            return true;
        }
        // Determine whether the type is a pointer.
        if(myType.IsPointer)
        {
            myElementType = "pointer";
            return true;
        }
        // Return false if the type is not a reference, array, or pointer
type.
        return false;
    }
}
public class Type_HasElementTypeImpl
{
    public static void Main()
    {
        try
        {
            int myInt = 0 ;
            int[] myArray = new int[5];
            MyTypeDelegator myType = new MyTypeDelegator(myArray.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            Console.WriteLine("\nDetermine whether a variable is an array,
pointer, or reference type.\n");
            if( myType.HasElementType)
                Console.WriteLine("The type of myArray is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myArray is not an array, pointer, or
reference type.");
            myType = new MyTypeDelegator(myInt.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            if( myType.HasElementType)
                Console.WriteLine("The type of myInt is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myInt is not an array, pointer, or
reference type.");
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
        }
    }
}
```

Remarks

If the current Type represents a generic type, or a type parameter in the definition of a generic type or generic method, this property always returns `false`.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsPointerImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsPrimitive Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is one of the primitive types.

C#

```
public bool IsPrimitive { get; }
```

Property Value

Boolean

`true` if the Type is one of the primitive types; otherwise, `false`.

Examples

The following example demonstrates the IsContextful, IsMarshalByRef, and `IsPrimitive` properties of the Type class. It checks whether the given type can be hosted in the context, whether it can be marshaled by reference, and whether the type is a primitive data type.

C#

```
using System;
using System.Runtime.Remoting.Contexts;

public class ContextBoundClass: ContextBoundObject
{
    public string Value = "The Value property.";
}

public class Example
{
    public static void Main()
    {
         // Determine whether the types can be hosted in a Context.
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
```

```
                            typeof(Example).Name,
typeof(Example).IsContextful);
         Console.WriteLine("The IsContextful property for the {0} type is
{1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsContextful);

         // Determine whether the types are marshalled by reference.
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(Example).Name,
typeof(Example).IsMarshalByRef);
         Console.WriteLine("The IsMarshalByRef property of {0} is {1}.",
                            typeof(ContextBoundClass).Name,
typeof(ContextBoundClass).IsMarshalByRef);

         // Determine whether the types are primitive datatypes.
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(int).Name, typeof(int).IsPrimitive);
         Console.WriteLine("{0} is a primitive data type: {1}.",
                            typeof(string).Name, typeof(string).IsPrimitive);
    }
}
// The example displays the following output:
//    The IsContextful property for the Example type is False.
//    The IsContextful property for the ContextBoundClass type is True.
//    The IsMarshalByRef property of Example is False.
//    The IsMarshalByRef property of ContextBoundClass is True.
//    Int32 is a primitive data type: True.
//    String is a primitive data type: False.
```

Remarks

The primitive types are Boolean, Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64, IntPtr, UIntPtr, Char, Double, and Single.

If the current Type represents a generic type, or a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Boolean
- Byte
- SByte
- Int16
- UInt16
- Int32
- UInt32
- Int64
- UInt64
- Char
- Double
- Single
- IsPrimitiveImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsPublic Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is declared public.

C#

```
public bool IsPublic { get; }
```

Property Value

Boolean

`true` if the Type is declared public and is not a nested type; otherwise, `false`.

Examples

The following example creates an instance of `MyTestClass`, checks for the `IsPublic` property, and displays the result.

C#

```
using System;

public class TestClass
{
}

public class Example
{
   public static void Main()
   {
      TestClass testClassInstance = new TestClass();
      // Get the type of myTestClassInstance.
      Type   testType = testClassInstance.GetType();
      // Get the IsPublic property of testClassInstance.
      bool isPublic = testType.IsPublic;
      Console.WriteLine("Is {0} public? {1}", testType.FullName, isPublic);
   }
```

```
}
// The example displays the following output:
//        Is TestClass public? True
```

For nested classes, ignore the results of `IsPublic` and `IsNotPublic` and pay attention only to the results of IsNestedPublic and IsNestedPrivate.

Remarks

Do not use with nested types; use IsNestedPublic instead.

If the current Type represents a type parameter of a generic type, this property returns `true`.

TypeAttributes.VisibilityMask selects the visibility attributes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSealed Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is declared sealed.

C#

```
public bool IsSealed { get; }
```

Property Value

Boolean

`true` if the Type is declared sealed; otherwise, `false`.

Examples

The following example creates an instance of a `sealed` class, checks for the `IsSealed` property, and displays the result.

C#

```
using System;

 public class Example
 {
     // Declare InnerClass as sealed.
     sealed public class InnerClass
     {
     }

     public static void Main()
     {
          InnerClass inner = new InnerClass();
          // Get the type of InnerClass.
          Type innerType = inner.GetType();
          // Get the IsSealed property of  innerClass.
          bool isSealed = innerType.IsSealed;
          Console.WriteLine("{0} is sealed: {1}.", innerType.FullName,
```

```
isSealed);
     }
}
// The example displays the following output:
//        Example+InnerClass is sealed: True.
```

Remarks

If the current Type represents a type parameter of a generic type, this property always returns `true`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSecurityCritical Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current type is security-critical or security-safe-critical at the current trust level, and therefore can perform critical operations.

```
C#
public virtual bool IsSecurityCritical { get; }
```

Property Value

Boolean

`true` if the current type is security-critical or security-safe-critical at the current trust level; `false` if it is transparent.

Remarks

The IsSecurityCritical, IsSecuritySafeCritical, and IsSecurityTransparent properties report the transparency level of the type at its current trust level, as determined by the common language runtime (CLR). The combinations of these properties are shown in the following table:

Expand table

Security level	IsSecurityCritical	IsSecuritySafeCritical	IsSecurityTransparent
Critical	`true`	`false`	`false`
Safe critical	`true`	`true`	`false`
Transparent	`false`	`false`	`true`

Using these properties is much simpler than examining the security annotations of an assembly and its types, checking the current trust level, and attempting to duplicate the

runtime's rules.

> **Important**
>
> For partial-trust assemblies, the value of this property depends on the current trust level of the assembly. If the assembly is loaded into a partially trusted application domain (for example, into a sandboxed application domain), then the runtime ignores the security annotations of the assembly. The assembly and all its types are treated as transparent. The runtime pays attention to the security annotations of a partial-trust assembly only when that assembly is loaded into a fully trusted application domain (for example, into the default application domain of a desktop application). By contrast, a trusted assembly (that is, a strong-named assembly that is installed in the global assembly cache) is always loaded with full trust regardless of the trust level of the application domain, so its current trust level is always fully trusted. You can determine the current trust levels of assemblies and application domains by using the **Assembly.IsFullyTrusted** and **AppDomain.IsFullyTrusted** properties.

For more information about reflection and transparency, see Security Considerations for Reflection. For information about transparency, see Security Changes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsSecuritySafeCritical
- IsSecurityTransparent
- Security Considerations for Reflection
- Security Changes in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsSecuritySafeCritical Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current type is security-safe-critical at the current trust level; that is, whether it can perform critical operations and can be accessed by transparent code.

C#

```
public virtual bool IsSecuritySafeCritical { get; }
```

Property Value

Boolean

`true` if the current type is security-safe-critical at the current trust level; `false` if it is security-critical or transparent.

Remarks

The IsSecurityCritical, IsSecuritySafeCritical, and IsSecurityTransparent properties report the transparency level of the type at its current trust level, as determined by the common language runtime (CLR). The combinations of these properties are shown in the following table:

Expand table

Security level	IsSecurityCritical	IsSecuritySafeCritical	IsSecurityTransparent
Critical	`true`	`false`	`false`
Safe critical	`true`	`true`	`false`
Transparent	`false`	`false`	`true`

Using these properties is much simpler than examining the security annotations of an assembly and its types, checking the current trust level, and attempting to duplicate the runtime's rules.

> **Important**
>
> For partial-trust assemblies, the value of this property depends on the current trust level of the assembly. If the assembly is loaded into a partially trusted application domain (for example, into a sandboxed application domain), then the runtime ignores the security annotations of the assembly. The assembly and all its types are treated as transparent. The runtime pays attention to the security annotations of a partial-trust assembly only when that assembly is loaded into a fully trusted application domain (for example, into the default application domain of a desktop application). By contrast, a trusted assembly (that is, a strong-named assembly that is installed in the global assembly cache) is always loaded with full trust regardless of the trust level of the application domain, so its current trust level is always fully trusted. You can determine the current trust levels of assemblies and application domains by using the **Assembly.IsFullyTrusted** and **AppDomain.IsFullyTrusted** properties.

For more information about reflection and transparency, see Security Considerations for Reflection. For information about transparency, see Security Changes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsSecurityCritical
- IsSecurityTransparent
- Security Considerations for Reflection
- Security Changes in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsSecurityTransparent Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current type is transparent at the current trust level, and therefore cannot perform critical operations.

```
C#
public virtual bool IsSecurityTransparent { get; }
```

Property Value

Boolean

`true` if the type is security-transparent at the current trust level; otherwise, `false`.

Remarks

If this property returns `true`, the IsSecurityCritical and IsSecuritySafeCritical properties return `false`.

The IsSecurityCritical, IsSecuritySafeCritical, and IsSecurityTransparent properties report the transparency level of the type at its current trust level, as determined by the common language runtime (CLR). Using these properties is much simpler than examining the security annotations of an assembly and its types, checking the current trust level, and attempting to duplicate the runtime's rules.

> **Important**
>
> For partial-trust assemblies, the value of this property depends on the current trust level of the assembly. If the assembly is loaded into a partially trusted application domain (for example, into a sandboxed application domain), then the runtime ignores the security annotations of the assembly. The assembly and all its types are treated as transparent. The runtime pays attention to the security annotations of a partial-trust assembly only when that assembly is loaded into a fully trusted

application domain (for example, into the default application domain of a desktop application). By contrast, a trusted assembly (that is, a strong-named assembly that is installed in the global assembly cache) is always loaded with full trust regardless of the trust level of the application domain, so its current trust level is always fully trusted. You can determine the current trust levels of assemblies and application domains by using the **Assembly.IsFullyTrusted** and **AppDomain.IsFullyTrusted** properties.

For more information about reflection and transparency, see Security Considerations for Reflection. For information about transparency, see Security Changes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsSecurityCritical
- IsSecuritySafeCritical
- Security Considerations for Reflection
- Security Changes in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSerializable Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Formatter-based serialization is obsolete and should not be used.

Gets a value indicating whether the Type is binary serializable.

C#

```
[System.Obsolete("Formatter-based serialization is obsolete and should not be used.", DiagnosticId="SYSLIB0050", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public virtual bool IsSerializable { get; }
```

Property Value

Boolean

`true` if the Type is binary serializable; otherwise, `false`.

Attributes ObsoleteAttribute

Examples

The following example creates an instance of `MyTestClass` class, sets the [Serializable] attribute, and checks the `IsSerializable` property for `true` or `false`.

C#

```
using System;
namespace SystemType
{
    public class MyClass
```

```
    {
        // Declare a public class with the [Serializable] attribute.
        [Serializable] public class MyTestClass
        {
        }
        public static void Main(string []args)
        {
            try
            {
                bool myBool = false;
                MyTestClass myTestClassInstance = new MyTestClass();
                // Get the type of myTestClassInstance.
                Type myType = myTestClassInstance.GetType();
                // Get the IsSerializable property of myTestClassInstance.
                myBool = myType.IsSerializable;
                Console.WriteLine("\nIs {0} serializable? {1}.",
myType.FullName, myBool.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("\nAn exception occurred: {0}",
e.Message);
            }
        }
    }
}
```

Remarks

Types that are defined in the .NET Standard are not marked with SerializableAttribute. Instead, each .NET implementation determines whether a type is binary serializable. At run time, you can use the IsSerializable property to determine whether that implementation supports binary serialization of an instance of the type. For more information and an example, see How to determine if a .NET Standard object is serializable.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- ISerializable

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSignatureType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is a signature type.

C#

```
public virtual bool IsSignatureType { get; }
```

Property Value

Boolean

`true` if the Type is a signature type; otherwise, `false`.

Remarks

A signature type is a restricted type that can be passed to the GetMethod method but does not support most of the other reflection functionality. For example, you can obtain the signature type instance corresponding to the generic method parameter by calling the MakeGenericMethodParameter method.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsSpecialName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the type has a name that requires special handling.

C#

```
public bool IsSpecialName { get; }
```

Property Value

Boolean

`true` if the type has a name that requires special handling; otherwise, `false`.

Remarks

Names that begin with or contain an underscore character (_), property accessors, and operator overloading methods are examples of types that might require special treatment by some compilers.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSZArray Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is an array type that can represent only a single-dimensional array with a zero lower bound.

```
C#
public virtual bool IsSZArray { get; }
```

Property Value

Boolean

`true` if the current Type is an array type that can represent only a single-dimensional array with a zero lower bound; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsTypeDefinition Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is a type definition.

C#

```
public virtual bool IsTypeDefinition { get; }
```

Property Value

Boolean

`true` if the current Type is a type definition; otherwise, `false`.

Remarks

A Type instance is a type definition if it represents the type directly defined in assembly, in contrast to types that are constructed on demand based on other types. For example, a primitive type, class, structure, or generic type definition is a type definition, but an array, reference, pointer, or instantiated generic type isn't.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsUnicodeClass Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the string format attribute `UnicodeClass` is selected for the Type.

```
C#
public bool IsUnicodeClass { get; }
```

Property Value

Boolean

`true` if the string format attribute `UnicodeClass` is selected for the Type; otherwise, `false`.

Remarks

The StringFormatMask is used to select the string format attributes. The string format attributes enhance interoperability by defining how strings should be interpreted.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsAnsiClass
- IsAutoClass

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsUnmanagedFunctionPointer Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the current Type is an unmanaged function pointer.

C#

```
public virtual bool IsUnmanagedFunctionPointer { get; }
```

Property Value

Boolean

`true` if the current Type is an unmanaged function pointer; otherwise, `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsValueType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type is a value type.

C#

```
public bool IsValueType { get; }
```

Property Value

Boolean

`true` if the Type is a value type; otherwise, `false`.

Examples

The following example creates a variable of type `MyEnum`, checks for the `IsValueType` property, and displays the result.

C#

```
using System;

// Declare an enum type.
enum NumEnum { One, Two }

public class Example
{

    public static void Main(string []args)
    {
        bool flag = false;
        NumEnum testEnum = NumEnum.One;
        // Get the type of testEnum.
        Type t = testEnum.GetType();
        // Get the IsValueType property of the testEnum variable.
        flag = t.IsValueType;
        Console.WriteLine("{0} is a value type: {1}", t.FullName, flag);
```

```
    }
}
// The example displays the following output:
//        NumEnum is a value type: True
```

Remarks

Value types are types that are represented as sequences of bits; value types are not classes or interfaces. Value types are referred to as "structs" in some programming languages. Enums are a special case of value types.

This property returns `false` for the ValueType class, because ValueType is not a value type itself. It is the base class for all value types, and therefore any value type can be assigned to it. This would not be possible if ValueType itself was a value type. Value types are boxed when they are assigned to a field of type ValueType.

This property returns `true` for enumerations, but not for the Enum type itself. For an example that demonstrates this behavior, see IsEnum.

This property is read-only.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsClass
- IsInterface
- ValueType
- IsValueTypeImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsVariableBoundArray Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the type is an array type that can represent a multi-dimensional array or an array with an arbitrary lower bound.

```
C#
public virtual bool IsVariableBoundArray { get; }
```

Property Value

Boolean

`true` if the current Type is an array type that can represent a multi-dimensional array or an array with an arbitrary lower bound; otherwise, `false`.

Remarks

If the value of this property is `true` for an array type, it can be used to create single-dimensional or multi-dimensional array instances with arbitrary lower bounds. Otherwise, the array type can only be used to create instances of single-dimensional arrays with a zero lower bound.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsVisible Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value indicating whether the Type can be accessed by code outside the assembly.

C#

```
public bool IsVisible { get; }
```

Property Value

Boolean

`true` if the current Type is a public type or a public nested type such that all the enclosing types are public; otherwise, `false`.

Examples

The following code example tests two classes, only one of which is visible outside the assembly.

C#

```
using System;

internal class InternalOnly
{
    public class Nested {}
}

public class Example
{
    public class Nested {}

    public static void Main()
    {
        Type t = typeof(InternalOnly.Nested);
        Console.WriteLine(
            "Is the {0} class visible outside the assembly? {1}",
```

```
            t.FullName,
            t.IsVisible
        );

        t = typeof(Example.Nested);
        Console.WriteLine(
            "Is the {0} class visible outside the assembly? {1}",
            t.FullName,
            t.IsVisible
        );
    }
}

/* This example produces the following output:

Is the InternalOnly+Nested class visible outside the assembly? False
Is the Example+Nested class visible outside the assembly? True
 */
```

Remarks

Use this property to determine whether a type is part of the public interface of a component assembly.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.MemberType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a MemberTypes value indicating that this member is a type or a nested type.

```
C#

public override System.Reflection.MemberTypes MemberType { get; }
```

Property Value

MemberTypes

A MemberTypes value indicating that this member is a type or a nested type.

Examples

The following code example shows the `MemberType` field as a parameter to the `GetMember` method:

```
C#

MemberInfo[] others = t.GetMember(mi.Name, mi.MemberType,
BindingFlags.Public |
BindingFlags.Static | BindingFlags.NonPublic | BindingFlags.Instance);
```

Remarks

This property overrides MemberInfo.MemberType. Therefore, when you examine a set of MemberInfo objects - for example, the array returned by GetMembers - the MemberType property returns MemberTypes.NestedType when a given member is a nested type.

If the current Type represents a constructed generic type, this property applies to the generic type definition from which the type was constructed. For example, if the current

Type represents `MyGenericType<int>` (`MyGenericType(Of Integer)` in Visual Basic), the value of this property is determined by `MyGenericType<T>`.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property always returns MemberTypes.TypeInfo.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberTypes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Module Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the module (the DLL) in which the current Type is defined.

C#

```
public abstract System.Reflection.Module Module { get; }
```

Property Value

Module

The module in which the current Type is defined.

Examples

This following example demonstrates a use of the Namespace and `Module` properties and the ToString method of Type.

C#

```
using System;

namespace MyNamespace
{
    class MyClass
    {
    }
}

public class Example
{
    public static void Main()
    {
         Type myType = typeof(MyNamespace.MyClass);
         Console.WriteLine("Displaying information about {0}:", myType);
         // Get the namespace of the myClass class.
         Console.WriteLine("   Namespace: {0}.", myType.Namespace);
```

```
            // Get the name of the module.
            Console.WriteLine("   Module: {0}.", myType.Module);
            // Get the fully qualified type name.
            Console.WriteLine("   Fully qualified name: {0}.",
myType.ToString());
        }
    }
// The example displays the following output:
//    Displaying information about MyNamespace.MyClass:
//       Namespace: MyNamespace.
//       Module: type_tostring.exe.
//       Fully qualified name: MyNamespace.MyClass.
```

Remarks

If the current Type represents a constructed generic type, this property returns the module in which the generic type definition was defined. For example, if you create an instance of `MyGenericStack<int>`, the Module property for the constructed type returns the module in which `MyGenericStack<T>` is defined.

Similarly, if the current Type represents a generic parameter `T`, this property returns the assembly that contains the generic type that defines `T`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Module

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.Namespace Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the namespace of the Type.

C#

```
public abstract string? Namespace { get; }
```

Property Value

String

The namespace of the Type; `null` if the current instance has no namespace or represents a generic parameter.

Examples

This following example demonstrates a use of the `Namespace` and Module properties and the ToString method of Type.

C#

```
using System;

namespace MyNamespace
{
    class MyClass
    {
    }
}

public class Example
{
    public static void Main()
    {
         Type myType = typeof(MyNamespace.MyClass);
         Console.WriteLine("Displaying information about {0}:", myType);
         // Get the namespace of the myClass class.
```

```
            Console.WriteLine("   Namespace: {0}.", myType.Namespace);
            // Get the name of the module.
            Console.WriteLine("   Module: {0}.", myType.Module);
            // Get the fully qualified type name.
            Console.WriteLine("   Fully qualified name: {0}.",
myType.ToString());
        }
}
// The example displays the following output:
//    Displaying information about MyNamespace.MyClass:
//       Namespace: MyNamespace.
//       Module: type_tostring.exe.
//       Fully qualified name: MyNamespace.MyClass.
```

Remarks

A namespace is a logical design-time naming convenience, used mainly to define scope in an application and organize classes and other types in a single hierarchical structure. From the viewpoint of the runtime, there are no namespaces.

If the current Type represents a constructed generic type, this property returns the namespace that contains the generic type definition. Similarly, if the current Type represents a generic parameter `T`, this property returns the namespace that contains the generic type definition that defines `T`.

If the current Type object represents a generic parameter and a generic type definition is not available, such as for a signature type returned by MakeGenericMethodParameter, this property returns `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String

- FullName
- AssemblyQualifiedName
- Specifying Fully Qualified Type Names

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.ReflectedType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the class object that was used to obtain this member.

C#

```
public override Type? ReflectedType { get; }
```

Property Value

Type

The `Type` object through which this Type object was obtained.

Examples

This example displays the reflected type of a nested class.

C#

```
using System;
using System.Reflection;

public abstract class MyClassA
{

    public abstract class MyClassB
    {
    }

    public static void Main(string[] args)
    {
        Console.WriteLine("Reflected type of MyClassB is {0}",
            typeof(MyClassB).ReflectedType); //outputs MyClassA, the
enclosing class
    }
}
```

Remarks

For Type objects, the value of this property is always the same as the value of the DeclaringType property.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberInfo
- DeclaringType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.StructLayoutAttribute Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a StructLayoutAttribute that describes the layout of the current type.

C#

```
public virtual System.Runtime.InteropServices.StructLayoutAttribute?
StructLayoutAttribute { get; }
```

Property Value

StructLayoutAttribute

Gets a StructLayoutAttribute that describes the gross layout features of the current type.

Exceptions

NotSupportedException

The invoked method is not supported in the base class.

Examples

The following code example first defines a class, a structure, and a structure with special layout attributes (the structures are nested within the class). The example then uses the StructLayoutAttribute property to obtain a StructLayoutAttribute for each type, and displays the properties of the attributes.

C#

```
using System;
using System.Runtime.InteropServices;

public class Example
{
    public static void Main()
    {
```

```
        DisplayLayoutAttribute(typeof(Example).StructLayoutAttribute);
        DisplayLayoutAttribute(typeof(Test1).StructLayoutAttribute);
        DisplayLayoutAttribute(typeof(Test2).StructLayoutAttribute);
    }

    private static void DisplayLayoutAttribute(StructLayoutAttribute sla)
    {
        Console.WriteLine("\r\nCharSet: "+sla.CharSet.ToString()+"\r\n
Pack: "+sla.Pack.ToString()+"\r\n   Size: "+sla.Size.ToString()+"\r\n
Value: "+sla.Value.ToString());
    }
    public struct Test1
    {
        public byte B1;
        public short S;
        public byte B2;
    }
    [StructLayout(LayoutKind.Explicit, Pack=1)] public struct Test2
    {
        [FieldOffset(0)] public byte B1;
        [FieldOffset(1)] public short S;
        [FieldOffset(3)] public byte B2;
    }
}
```

Remarks

StructLayoutAttribute is not returned by the GetCustomAttributes method. Instead, use this property to get it.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- StructLayoutAttribute

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.TypeHandle Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the handle for the current Type.

C#

```
public virtual RuntimeTypeHandle TypeHandle { get; }
```

Property Value

RuntimeTypeHandle

The handle for the current Type.

Exceptions

NotSupportedException

The .NET Compact Framework does not currently support this property.

Examples

The following example returns the handle of the corresponding type and passes the handle to a method that gets the type from the handle and displays it.

C#

```
using System;
using System.Reflection;
class MyClass
{
    public int myField = 10;
}

class Type_TypeHandle
{
    public static void Main()
    {
```

```
        try
        {
            MyClass myClass = new MyClass();

            // Get the type of MyClass.
            Type myClassType = myClass.GetType();

            // Get the runtime handle of MyClass.
            RuntimeTypeHandle myClassHandle = myClassType.TypeHandle;

            DisplayTypeHandle(myClassHandle);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: {0}", e.Message );
        }
    }

    public static void DisplayTypeHandle(RuntimeTypeHandle myTypeHandle)
    {
        // Get the type from the handle.
        Type myType = Type.GetTypeFromHandle(myTypeHandle);
        // Display the type.
        Console.WriteLine("\nDisplaying the type from the handle:\n");
        Console.WriteLine("The type is {0}.", myType.ToString());
    }
}
```

Remarks

`TypeHandle` encapsulates a pointer to an internal data structure that represents the type. This handle is unique during the process lifetime. The handle is valid only in the application domain in which it was obtained.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- RuntimeTypeHandle
- GetTypeHandle(Object)
- GetTypeFromHandle(RuntimeTypeHandle)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.TypeInitializer Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the initializer for the type.

C#

```
public System.Reflection.ConstructorInfo? TypeInitializer { get; }
```

Property Value

ConstructorInfo

An object that contains the name of the class constructor for the Type.

Remarks

Class initializers are also available through the FindMembers method, or through overloads of the GetMember, GetMembers, GetConstructor, and GetConstructors methods that take BindingFlags as a parameter.

If the current Type represents a type parameter in the definition of a generic type or generic method, this property returns `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ConstructorInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.UnderlyingSystemType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates the type provided by the common language runtime that represents this type.

C#

```
public abstract Type UnderlyingSystemType { get; }
```

Property Value

Type

The underlying system type for the Type.

Implements

UnderlyingSystemType

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- BaseType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Object or Type.

Overloads

Expand table

Equals(Type)	Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Type.
Equals(Object)	Determines if the underlying system type of the current Type object is the same as the underlying system type of the specified Object.

Equals(Type)

Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Type.

C#

```
public virtual bool Equals (Type? o);
```

Parameters

o Type

The object whose underlying system type is to be compared with the underlying system type of the current Type.

Returns

Boolean

`true` if the underlying system type of `o` is the same as the underlying system type of the current Type; otherwise, `false`.

Examples

The following example uses `Equals` to compare two types.

C#

```
using System;
using System.Reflection;

class Example
{
    public static void Main()
    {

        Type a = typeof(System.String);
        Type b = typeof(System.Int32);

        Console.WriteLine("{0} == {1}: {2}", a, b, a.Equals(b));

        // The Type objects in a and b are not equal,
        // because they represent different types.

        a = typeof(Example);
        b = new Example().GetType();

        Console.WriteLine("{0} is equal to {1}: {2}", a, b, a.Equals(b));

        // The Type objects in a and b are equal,
        // because they both represent type Example.

        b = typeof(Type);

        Console.WriteLine("typeof({0}).Equals(typeof({1})): {2}", a, b,
a.Equals(b));

        // The Type objects in a and b are not equal,
        // because variable a represents type Example
        // and variable b represents type Type.

        //Console.ReadLine();
    }
}

//
/* This code example produces the following output:
    System.String == System.Int32: False
    Example is equal to Example: True
```

```
    typeof(Example).Equals(typeof(System.Type)): False
*/
```

See also

- UnderlyingSystemType

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Object)

Determines if the underlying system type of the current Type object is the same as the underlying system type of the specified Object.

C#

```
public override bool Equals (object? o);
```

Parameters

o Object

The object whose underlying system type is to be compared with the underlying system type of the current Type. For the comparison to succeed, `o` must be able to be cast or converted to an object of type Type.

Returns

Boolean

`true` if the underlying system type of `o` is the same as the underlying system type of the current Type; otherwise, `false`. This method also returns `false` if:

- `o` is `null`.
- `o` cannot be cast or converted to a Type object.

Examples

The following example uses Equals(Object) to compare various Type object instances with various Object instances.

C#

```
using System;
using System.Collections.Generic;
using System.Reflection;

public class Example
{
   public static void Main()
   {
      Type t =typeof(int);
      Object obj1 = typeof(int).GetTypeInfo();
      IsEqualTo(t, obj1);

      Object obj2 = typeof(String);
      IsEqualTo(t, obj2);

      t = typeof(Object);
      Object obj3 = typeof(Object);
      IsEqualTo(t, obj3);

      t = typeof(List<>);
      Object obj4 = (new List<String>()).GetType();
      IsEqualTo(t, obj4);

      t = typeof(Type);
      Object obj5 = null;
      IsEqualTo(t, obj5);
   }

   private static void IsEqualTo(Type t, Object inst)
   {
      Type t2 = inst as Type;
      if (t2 != null)
         Console.WriteLine("{0} = {1}: {2}", t.Name, t2.Name,
                           t.Equals(t2));
      else
         Console.WriteLine("Cannot cast the argument to a type.");
```

```
        Console.WriteLine();
    }
}
// The example displays the following output:
//       Int32 = Int32: True
//
//       Int32 = String: False
//
//       Object = Object: True
//
//       List`1 = List`1: False
//
//       Cannot cast the argument to a type.
```

Two things are particularly worth noting about the example:

- The comparison of a Type object that represents an integer with a TypeInfo object that represents an integer return `true` because TypeInfo is derived from Type.
- The comparison of a Type object that represents a IList<T> object (an open generic type) with a `List(Of String)` object (a closed generic type) returns `false`.

Remarks

This method overrides Object.Equals. It casts `o` to an object of type Type and calls the Type.Equals(Type) method.

See also

- UnderlyingSystemType

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.FindInterfaces(TypeFilter, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an array of Type objects representing a filtered list of interfaces implemented or inherited by the current Type.

C#

```
public virtual Type[] FindInterfaces (System.Reflection.TypeFilter filter,
object? filterCriteria);
```

Parameters

`filter` TypeFilter

The delegate that compares the interfaces against `filterCriteria`.

`filterCriteria` Object

The search criteria that determines whether an interface should be included in the returned array.

Returns

Type[]

An array of Type objects representing a filtered list of the interfaces implemented or inherited by the current Type, or an empty array if no interfaces matching the filter are implemented or inherited by the current Type.

Exceptions

ArgumentNullException

`filter` is `null`.

TargetInvocationException

A static initializer is invoked and throws an exception.

Examples

The following example finds the specified interface implemented or inherited by the specified type, and then displays the interface names.

C#

```
using System;
using System.Xml;
using System.Reflection;

public class MyFindInterfacesSample
{
    public static void Main()
    {
        try
        {
            XmlDocument myXMLDoc = new XmlDocument();
            myXMLDoc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +
                "<title>Pride And Prejudice</title>" + "</book>");
            Type myType = myXMLDoc.GetType();

            // Specify the TypeFilter delegate that compares the
            // interfaces against filter criteria.
            TypeFilter myFilter = new TypeFilter(MyInterfaceFilter);
            String[] myInterfaceList = new String[2]
                {"System.Collections.IEnumerable",
                "System.Collections.ICollection"};
            for(int index=0; index < myInterfaceList.Length; index++)
            {
                Type[] myInterfaces = myType.FindInterfaces(myFilter,
                    myInterfaceList[index]);
                if (myInterfaces.Length > 0)
                {
                    Console.WriteLine("\n{0} implements the interface {1}.",
                        myType, myInterfaceList[index]);
                    for(int j =0;j < myInterfaces.Length;j++)
                        Console.WriteLine("Interfaces supported: {0}.",
                            myInterfaces[j].ToString());
                }
                else
                    Console.WriteLine(
                        "\n{0} does not implement the interface {1}.",
                        myType,myInterfaceList[index]);
            }
        }
        catch(ArgumentNullException e)
        {
```

```
            Console.WriteLine("ArgumentNullException: " + e.Message);
        }
        catch(TargetInvocationException e)
        {
            Console.WriteLine("TargetInvocationException: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
    }

    public static bool MyInterfaceFilter(Type typeObj,Object criteriaObj)
    {
        if(typeObj.ToString() == criteriaObj.ToString())
            return true;
        else
            return false;
    }
}
```

Remarks

This method can be overridden by a derived class.

The Module.FilterTypeName and Module.FilterTypeNameIgnoreCase delegates supplied by the System.Reflection.Module class may also be used, in lieu of the System.Reflection.TypeFilter delegate.

All of the interfaces implemented by this class are considered during the search, whether declared by a base class or this class itself.

This method searches the base class hierarchy, returning each of the matching interfaces each class implements as well as all the matching interfaces each of those interfaces implements (that is, the transitive closure of the matching interfaces is returned). No duplicate interfaces are returned.

If the current Type represents a type parameter in the definition of a generic type or generic method, FindInterfaces searches all the interfaces declared in the constraints on the type parameter, and all interfaces inherited through the interfaces declared in the constraints. If the current Type represents a type argument of a generic type, FindInterfaces searches all the interfaces implemented by the type, whether or not they match constraints.

ⓘ **Note**

FindInterfaces can return generic interfaces, even on types that are not generic. For example, a nongeneric type might implement `IEnumerable<int>` (`IEnumerable(Of Integer)` in Visual Basic).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Module
- TypeFilter
- GetInterface(String)
- GetInterfaces()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.FindMembers(MemberTypes, BindingFlags, MemberFilter, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a filtered array of MemberInfo objects of the specified member type.

C#

```
public virtual System.Reflection.MemberInfo[] FindMembers
(System.Reflection.MemberTypes memberType, System.Reflection.BindingFlags
bindingAttr, System.Reflection.MemberFilter? filter, object?
filterCriteria);
```

Parameters

`memberType` MemberTypes

A bitwise combination of the enumeration values that indicates the type of member to search for.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`filter` MemberFilter

The delegate that does the comparisons, returning `true` if the member currently being inspected matches the `filterCriteria` and `false` otherwise.

`filterCriteria` Object

The search criteria that determines whether a member is returned in the array of `MemberInfo` objects.

The fields of `FieldAttributes`, `MethodAttributes`, and `MethodImplAttributes` can be used in conjunction with the `FilterAttribute` delegate supplied by this class.

Returns

MemberInfo[]

A filtered array of MemberInfo objects of the specified member type.

-or-

An empty array if the current Type does not have members of type `memberType` that match the filter criteria.

Exceptions

ArgumentNullException

`filter` is `null`.

Examples

The following example finds all the members in a class that match the specified search criteria, and then displays the matched members.

C#

```
using System;
using System.Reflection;

class MyFindMembersClass
{
    public static void Main()
    {
        Object objTest = new Object();
        Type objType = objTest.GetType ();
        MemberInfo[] arrayMemberInfo;
        try
        {
            //Find all static or public methods in the Object class that
match the specified name.
            arrayMemberInfo = objType.FindMembers(MemberTypes.Method,
                BindingFlags.Public | BindingFlags.Static|
BindingFlags.Instance,
                new MemberFilter(DelegateToSearchCriteria),
```

```
                "ReferenceEquals");

            for(int index=0;index < arrayMemberInfo.Length ;index++)
                Console.WriteLine ("Result of FindMembers -\t"+
arrayMemberInfo[index].ToString() +"\n");
        }
        catch (Exception e)
        {
            Console.WriteLine ("Exception : " + e.ToString() );
        }
    }
    public static bool DelegateToSearchCriteria(MemberInfo objMemberInfo,
Object objSearch)
    {
        // Compare the name of the member function with the filter criteria.
        if(objMemberInfo.Name.ToString() == objSearch.ToString())
            return true;
        else
            return false;
    }
}
/* The example produces the following output:

Result of FindMembers - Boolean ReferenceEquals(System.Object,
System.Object)
*/
```

Remarks

This method can be overridden by a derived class.

Members include properties, methods, fields, events, and so on.

For the `FindMembers` method to successfully retrieve member information, the `bindingAttr` argument must include at least one of BindingFlags.Instance and BindingFlags.Static, along with at least one of BindingFlags.NonPublic and BindingFlags.Public.

The following BindingFlags filter flags can be used to define which members to include in the search:

- Specify `BindingFlags.Instance` to include instance members in the search.
- Specify `BindingFlags.Static` to include static members in the search.
- Specify `BindingFlags.Public` to include public members in the search.

- Specify `BindingFlags.NonPublic` to include non-public members (that is, private, internal, and protected members) in the search.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

See System.Reflection.BindingFlags for more information.

To get the class initializer (static constructor) using this method, you must specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

If the current Type represents a type parameter of a generic type or generic method, FindMembers processes any members declared by the class constraint and the interface constraints of the type parameter.

The `filter` argument can be a custom delegate of type MemberFilter, or it can be one of the following predefined delegates:

- Type.FilterAttribute, which uses aFieldAttributes, MethodAttributes, or MethodImplAttributes bitmask as the `filterCriteria` value.

- Type.FilterName, which performs a case-sensitive comparison of each member name with the string passed to `filterCriteria`.

- Type.FilterNameIgnoreCase, which performs a case-insensitive comparison of each member name with the string passed to `filterCriteria`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberInfo

- BindingFlags
- DefaultBinder
- GetMember(String)
- GetMembers()
- GetDefaultMembers()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetArrayRank Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of dimensions in an array.

C#

```
public virtual int GetArrayRank ();
```

Returns

Int32

An integer that contains the number of dimensions in the current type.

Exceptions

NotSupportedException

The functionality of this method is unsupported in the base class and must be implemented in a derived class instead.

ArgumentException

The current type is not an array.

Examples

The following example displays the number of dimensions in an array.

C#

```
using System;

class MyArrayRankSample
{
    public static void Main()
    {
        try
```

```
        {
            int[,,] myArray = new int[,,] {{{12,2,35},{300,78,33}},
{{92,42,135},{30,7,3}}};
            Type myType = myArray.GetType();

            Console.WriteLine("Contents of myArray: {{{12,2,35},
{300,78,33}},{{92,42,135},{30,7,3}}}");
            Console.WriteLine("myArray has {0} dimensions.",
myType.GetArrayRank());
        }
        catch(NotSupportedException e)
        {
            Console.WriteLine("NotSupportedException raised.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception raised.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
    }
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Array
- Rank

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.GetAttributeFlagsImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the Attributes property and gets a bitwise combination of enumeration values that indicate the attributes associated with the Type.

C#

```
protected abstract System.Reflection.TypeAttributes GetAttributeFlagsImpl
();
```

Returns

TypeAttributes

A TypeAttributes object representing the attribute set of the Type.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Attributes
- TypeAttributes

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetConstructor Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific constructor of the current Type.

Overloads

Expand table

GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetConstructor(BindingFlags, Binder, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetConstructor(BindingFlags, Type[])	Searches for a constructor whose parameters match the specified argument types, using the specified binding constraints.
GetConstructor(Type[])	Searches for a public instance constructor whose parameters match the types in the specified array.

GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

```
C#
```

```
public System.Reflection.ConstructorInfo? GetConstructor
(System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, System.Reflection.CallingConventions callConvention, Type[]
types, System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and the stack is cleaned up.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the constructor to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a constructor that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

ConstructorInfo

An object representing the constructor that matches the specified requirements, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

Examples

The following example obtains the type of `MyClass`, gets the ConstructorInfo object, and displays the constructor signature.

C#

```
using System;
using System.Reflection;
using System.Security;

public class MyClass1
{
    public MyClass1(int i){}
    public static void Main()
    {
        try
        {
            Type  myType = typeof(MyClass1);
            Type[] types = new Type[1];
            types[0] = typeof(int);
```

```
            // Get the public instance constructor that takes an integer
parameter.
            ConstructorInfo constructorInfoObj = myType.GetConstructor(
                BindingFlags.Instance | BindingFlags.Public, null,
                CallingConventions.HasThis, types, null);
            if(constructorInfoObj != null)
            {
                Console.WriteLine("The constructor of MyClass1 that is a
public " +
                    "instance method and takes an integer as a parameter
is: ");
                Console.WriteLine(constructorInfoObj.ToString());
            }
            else
            {
                Console.WriteLine("The constructor of MyClass1 that is a
public instance " +
                    "method and takes an integer as a parameter is not
available.");
            }
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException: " + e.Message);
        }
        catch(ArgumentException e)
        {
            Console.WriteLine("ArgumentException: " + e.Message);
        }
        catch(SecurityException e)
        {
            Console.WriteLine("SecurityException: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
    }
}
```

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

If an exact match does not exist, the `binder` will attempt to coerce the parameter types specified in the `types` array in order to select a match. If the `binder` is unable

to select a match, then `null` is returned.

The following BindingFlags filter flags can be used to define which constructors to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public constructors in the search.
- Specify `BindingFlags.NonPublic` to include non-public constructors (that is, private, internal, and protected constructors) in the search.

See System.Reflection.BindingFlags for more information.

To get the class initializer (static constructor) using this method, you must specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the ConstructorInfo with the type parameters replaced by the appropriate type arguments. If the current Type represents a type parameter in the definition of a generic type or generic method, this method always returns `null`.

See also

- ConstructorInfo
- BindingFlags
- Binder
- DefaultBinder
- CallingConventions
- ParameterModifier
- GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetConstructor(BindingFlags, Binder, Type[], ParameterModifier[])

Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints.

C#

```
public System.Reflection.ConstructorInfo? GetConstructor
(System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, Type[] types, System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the constructor to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a constructor that takes no parameters.

-or-

EmptyTypes.

modifiers ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the parameter type array. The default binder does not process this parameter.

Returns

ConstructorInfo

A ConstructorInfo object representing the constructor that matches the specified requirements, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

Examples

The following example obtains the type of `MyClass`, gets the ConstructorInfo object, and displays the constructor signature.

C#

```
using System;
using System.Reflection;
using System.Security;

public class MyClass1
{
    public MyClass1(int i){}
    public static void Main()
    {
        try
        {
            Type myType = typeof(MyClass1);
            Type[] types = new Type[1];
            types[0] = typeof(int);
            // Get the constructor that is public and takes an integer
parameter.
            ConstructorInfo constructorInfoObj = myType.GetConstructor(
                BindingFlags.Instance | BindingFlags.Public, null, types,
null);
            if (constructorInfoObj != null )
            {
                Console.WriteLine("The constructor of MyClass1 that is
public " +
                    "and takes an integer as a parameter is:");
                Console.WriteLine(constructorInfoObj.ToString());
            }
            else
            {
                Console.WriteLine("The constructor of the MyClass1 that
is public " +
                    "and takes an integer as a parameter is not
available.");
            }
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException: " + e.Message);
        }
        catch(ArgumentException e)
        {
            Console.WriteLine("ArgumentException: " + e.Message);
        }
        catch(SecurityException e)
        {
            Console.WriteLine("SecurityException: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
```

```
        }
    }
}
```

Remarks

If an exact match does not exist, the `binder` will attempt to coerce the parameter types specified in the `types` array in order to select a match. If the `binder` is unable to select a match, then `null` is returned.

The following BindingFlags filter flags can be used to define which constructors to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public constructors in the search.
- Specify `BindingFlags.NonPublic` to include non-public constructors (that is, private, internal, and protected constructors) in the search.

See System.Reflection.BindingFlags for more information.

To get the class initializer (static constructor) using this method overload, you must specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the ConstructorInfo with the type parameters replaced by the appropriate type arguments. If the current Type represents a type parameter in the definition of a generic type or generic method, this method always returns `null`.

See also

- ConstructorInfo
- BindingFlags

- Binder
- DefaultBinder
- ParameterModifier
- GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetConstructor(BindingFlags, Type[])

Searches for a constructor whose parameters match the specified argument types, using the specified binding constraints.

C#

```
public System.Reflection.ConstructorInfo? GetConstructor
(System.Reflection.BindingFlags bindingAttr, Type[] types);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted. -or- Default to return `null`.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the constructor to get. -or- An empty array of the type Type (that is, Type[] types = Array.Empty{Type}()) to get a constructor that takes no parameters. -or- EmptyTypes.

Returns

ConstructorInfo

A ConstructorInfo object representing the constructor that matches the specified requirements, if found; otherwise, `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

GetConstructor(Type[])

Searches for a public instance constructor whose parameters match the types in the specified array.

C#

```
public System.Reflection.ConstructorInfo? GetConstructor (Type[] types);
```

Parameters

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the desired constructor.

-or-

An empty array of Type objects, to get a constructor that takes no parameters. Such an empty array is provided by the `static` field EmptyTypes.

Returns

ConstructorInfo

An object representing the public instance constructor whose parameters match the types in the parameter type array, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

Examples

The following example obtains the type of `MyClass`, gets the ConstructorInfo object, and displays the constructor signature.

C#

```
using System;
using System.Reflection;
using System.Security;

public class MyClass1
{
    public MyClass1(){}
    public MyClass1(int i){}

    public static void Main()
    {
        try
        {
            Type myType = typeof(MyClass1);
            Type[] types = new Type[1];
            types[0] = typeof(int);
            // Get the constructor that takes an integer as a parameter.
            ConstructorInfo constructorInfoObj =
myType.GetConstructor(types);
            if (constructorInfoObj != null)
            {
                Console.WriteLine("The constructor of MyClass1 that takes
an " +
                    "integer as a parameter is: ");
                Console.WriteLine(constructorInfoObj.ToString());
            }
            else
            {
                Console.WriteLine("The constructor of MyClass1 that takes
an integer " +
```

```
                    "as a parameter is not available.");
            }
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception caught.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
    }
}
```

Remarks

This method overload looks for public instance constructors and cannot be used to obtain a class initializer (static constructor). To get a class initializer, use an overload that takes BindingFlags, and specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

If the requested constructor is non-public, this method returns `null`.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the ConstructorInfo with the type parameters replaced by the appropriate type arguments. If the current Type represents a type parameter in the definition of a generic type or generic method, this method always returns `null`.

See also

- ConstructorInfo
- DefaultBinder
- GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetConstructorImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

C#

```
protected abstract System.Reflection.ConstructorInfo? GetConstructorImpl
(System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, System.Reflection.CallingConventions callConvention, Type[] types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and

the stack is cleaned up.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the constructor to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a constructor that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

ConstructorInfo

A ConstructorInfo object representing the constructor that matches the specified requirements, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

NotSupportedException

The current type is a TypeBuilder or GenericTypeParameterBuilder.

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

If an exact match does not exist, the `binder` will attempt to coerce the parameter types specified in the `types` array in order to select a match. If the `binder` is unable to select a match, then `null` is returned.

The following BindingFlags filter flags can be used to define which constructors to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public constructors in the search.
- Specify `BindingFlags.NonPublic` to include non-public constructors (that is, private, internal, and protected constructors) in the search.

See System.Reflection.BindingFlags for more information.

This method implements GetConstructor.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ConstructorInfo
- BindingFlags
- Binder

- DefaultBinder
- CallingConventions
- ParameterModifier
- GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetConstructors Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the constructors of the current Type.

Overloads

Expand table

GetConstructors()	Returns all the public constructors defined for the current Type.
GetConstructors(BindingFlags)	When overridden in a derived class, searches for the constructors defined for the current Type, using the specified `BindingFlags`.

Examples

This example shows the output of the GetConstructors() overload from a class that has two instance constructors and one static constructor.

C#

```
using System;
using System.Reflection;

public class t {
    public t() {}
    static t() {}
    public t(int i) {}

    public static void Main() {
        ConstructorInfo[] p = typeof(t).GetConstructors();
        Console.WriteLine(p.Length);

        for (int i=0;i<p.Length;i++) {
            Console.WriteLine(p[i].IsStatic);
        }
    }
}
```

The output of this code is:

Output

```
2
False
False
```

Because the GetConstructors overload uses only Public and Instance, the static constructor is neither counted by the `for` expression nor evaluated by `IsStatic`.

To find static constructors, use the GetConstructors overload, and pass it the combination (logical OR) of BindingFlags.Public, BindingFlags.Static, BindingFlags.NonPublic, BindingFlags.Instance, as shown in the following code example:

C#

```
using System;
using System.Reflection;

public class t {
    public t() {}
    static t() {}
    public t(int i) {}

    public static void Main() {
        ConstructorInfo[] p = typeof(t).GetConstructors(
           BindingFlags.Public | BindingFlags.Static |
           BindingFlags.NonPublic | BindingFlags.Instance);
        Console.WriteLine(p.Length);

        for (int i=0;i<p.Length;i++) {
            Console.WriteLine(p[i].IsStatic);
        }
    }
}
```

Now the output is:

Output

```
3
False
True
False
```

GetConstructors()

Returns all the public constructors defined for the current Type.

```
C#

public System.Reflection.ConstructorInfo[] GetConstructors ();
```

Returns

ConstructorInfo[]

An array of ConstructorInfo objects representing all the public instance constructors defined for the current Type, but not including the type initializer (static constructor). If no public instance constructors are defined for the current Type, or if the current Type represents a type parameter in the definition of a generic type or generic method, an empty array of type ConstructorInfo is returned.

Remarks

In .NET 6 and earlier versions, the GetConstructors method does not return constructors in a particular order, such as declaration order. Your code must not depend on the order in which constructors are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No

Member Type	Static	Non-Static
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

This method overload calls the GetConstructors(BindingFlags) method overload, with BindingFlags.Public | BindingFlags.Instance (BindingFlags.Public `Or` BindingFlags.Instance in Visual Basic). It will not find class initializers (static constructor). To find class initializers, use an overload that takes BindingFlags, and specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

If the current Type represents a constructed generic type, this method returns the ConstructorInfo objects with the type parameters replaced by the appropriate type arguments. For example, if class `C<T>` has a constructor `C(T t1)` (`Sub New(ByVal t1 As T)` in Visual Basic), calling GetConstructors on `C<int>` returns a ConstructorInfo that represents `C(int t1)` in C# (`Sub New(ByVal t1 As Integer)` in Visual Basic).

If the current Type represents a generic type parameter, the GetConstructors method returns an empty array.

See also

- ConstructorInfo
- GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetConstructors(BindingFlags)

When overridden in a derived class, searches for the constructors defined for the current Type, using the specified `BindingFlags`.

C#

```
public abstract System.Reflection.ConstructorInfo[] GetConstructors
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

ConstructorInfo[]

An array of ConstructorInfo objects representing all constructors defined for the current Type that match the specified binding constraints, including the type initializer if it is defined. Returns an empty array of type ConstructorInfo if no constructors are defined for the current Type, if none of the defined constructors match the binding constraints, or if the current Type represents a type parameter in the definition of a generic type or generic method.

Remarks

`bindingAttr` can be used to specify whether to return only public constructors or both public and non-public constructors.

The following BindingFlags filter flags can be used to define which constructors to include in the search:

- Specify `BindingFlags.Static` along with `BindingFlags.NonPublic` to retrieve the class initializer (static constructor). You can also get the class initializer using the TypeInitializer property.
- Specify `BindingFlags.Instance` along with one or both of `BindingFlags.Public` and `BindingFlags.NonPublic` to retrieve instance constructors.

See System.Reflection.BindingFlags for more information.

In .NET 6 and earlier versions, the GetConstructors method does not return constructors in a particular order, such as declaration order. Your code must not depend on the order in which constructors are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

If the current Type represents a constructed generic type, this method returns the ConstructorInfo objects with the type parameters replaced by the appropriate type arguments. For example, if class `C<T>` has a constructor `C(T t1)` (`Sub New(ByVal t1 As T)` in Visual Basic), calling GetConstructors on `C<int>` returns a ConstructorInfo that represents `C(int t1)` in C# (`Sub New(ByVal t1 As Integer)` in Visual Basic).

If the current Type represents a generic type parameter, the GetConstructors method returns an empty array.

See also

- ConstructorInfo
- BindingFlags
- DefaultBinder
- GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetDefaultMembers Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for the members defined for the current Type whose DefaultMemberAttribute is set.

C#

```
public virtual System.Reflection.MemberInfo[] GetDefaultMembers ();
```

Returns

MemberInfo[]

An array of MemberInfo objects representing all default members of the current Type.

-or-

An empty array of type MemberInfo, if the current Type does not have default members.

Examples

The following example obtains the default member information of `MyClass` and displays the default members.

C#

```
using System;
using System.Reflection;
using System.IO;

[DefaultMemberAttribute("Age")]
public class MyClass
{
    public void Name(String s) {}
    public int Age
    {
        get
```

```
        {
            return 20;
        }
    }
    public static void Main()
    {
        try
        {
            Type  myType = typeof(MyClass);
            MemberInfo[] memberInfoArray = myType.GetDefaultMembers();
            if (memberInfoArray.Length > 0)
            {
                foreach(MemberInfo memberInfoObj in memberInfoArray)
                {
                    Console.WriteLine("The default member name is: " +
memberInfoObj.ToString());
                }
            }
            else
            {
                Console.WriteLine("No default members are available.");
            }
        }
        catch(InvalidOperationException e)
        {
            Console.WriteLine("InvalidOperationException: " + e.Message);
        }
        catch(IOException e)
        {
            Console.WriteLine("IOException: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
    }
}
```

Remarks

The GetDefaultMembers method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies.

This method can be overridden by a derived class.

Members include properties, methods, fields, events, and so on.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the MemberInfo objects with the type parameters replaced by the appropriate type arguments. For example, if class `C<T>` has a property `P` that returns `T`, calling GetDefaultMembers on `C<int>` returns `int P` in C# (`Property P As Integer` in Visual Basic).

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- MemberInfo
- DefaultMemberAttribute
- GetMember(String)
- GetMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetElementType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the Type of the object encompassed or referred to by the current array, pointer or reference type.

C#

```
public abstract Type? GetElementType ();
```

Returns

Type

The Type of the object encompassed or referred to by the current array, pointer, or reference type, or `null` if the current Type is not an array or a pointer, or is not passed by reference, or represents a generic type or a type parameter in the definition of a generic type or generic method.

Examples

The following example demonstrates using the `GetElementType` method.

C#

```
using System;
class TestGetElementType
{
    public static void Main()
    {
        int[] array = {1,2,3};
        Type t = array.GetType();
        Type t2 = t.GetElementType();
        Console.WriteLine("The element type of {0} is {1}.",array,
t2.ToString());
        TestGetElementType newMe = new TestGetElementType();
        t = newMe.GetType();
        t2 = t.GetElementType();
```

```
        Console.WriteLine("The element type of {0} is {1}.", newMe,
t2==null? "null" : t2.ToString());
    }
}

/* This code produces the following output:

The element type of System.Int32[] is System.Int32.
The element type of TestGetElementType is null.
 */
```

Remarks

This method returns `null` for the Array class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- HasElementType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetEnumName(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the name of the constant that has the specified value, for the current enumeration type.

```
C#
public virtual string? GetEnumName (object value);
```

Parameters

`value` Object

The value whose name is to be retrieved.

Returns

String

The name of the member of the current enumeration type that has the specified value, or `null` if no such constant is found.

Exceptions

ArgumentException

The current type is not an enumeration.

-or-

`value` is neither of the current type nor does it have the same underlying type as the current type.

ArgumentNullException

`value` is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetEnumNames Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the names of the members of the current enumeration type.

C#

```
public virtual string[] GetEnumNames ();
```

Returns

String[]

An array that contains the names of the members of the enumeration.

Exceptions

ArgumentException

The current type is not an enumeration.

Remarks

The elements of the return value array are sorted by the binary values (that is, the unsigned values) of the enumerated constants. If the array contains enumerated constants with the same value, the order of their corresponding names is unspecified.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetEnumUnderlyingType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the underlying type of the current enumeration type.

C#

```
public virtual Type GetEnumUnderlyingType ();
```

Returns

Type

The underlying type of the current enumeration.

Exceptions

ArgumentException

The current type is not an enumeration.

-or-

The enumeration type is not valid, because it contains more than one instance field.

Remarks

By default, the underlying type of an enumeration in C# and Visual Basic is Int32. Other integer types can be specified.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetEnumValues Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an array of the values of the constants in the current enumeration type.

C#

```
public virtual Array GetEnumValues ();
```

Returns

Array

An array that contains the values. The elements of the array are sorted by the binary values (that is, the unsigned values) of the enumeration constants.

Exceptions

ArgumentException

The current type is not an enumeration.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetEnumValuesAsUnderlyingType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Retrieves an array of the values of the underlying type constants of this enumeration type.

```
C#
public virtual Array GetEnumValuesAsUnderlyingType ();
```

Returns

Array

An array that contains the values of the underlying type constants in this enumeration type.

Exceptions

ArgumentException

This type is not an enumeration type.

Remarks

You can use this method to get enumeration values when it's hard to create an array of the enumeration type. For example, you might use this method for the MetadataLoadContext enumeration or on a platform where run-time code generation is not available.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetEvent Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific event declared or inherited by the current Type.

Overloads

Expand table

GetEvent(String, BindingFlags)	When overridden in a derived class, returns the EventInfo object representing the specified event, using the specified binding constraints.
GetEvent(String)	Returns the EventInfo object representing the specified public event.

GetEvent(String, BindingFlags)

When overridden in a derived class, returns the EventInfo object representing the specified event, using the specified binding constraints.

C#

```
public abstract System.Reflection.EventInfo? GetEvent (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

name String

The string containing the name of an event which is declared or inherited by the current Type.

bindingAttr BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

EventInfo

The object representing the specified event that is declared or inherited by the current Type, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following code example uses the GetEvent(String, BindingFlags) method to search a type for a public or non-public event named "Click" that is not `static` (`Shared` in Visual Basic).

C#

```
using System;
using System.Reflection;
using System.Security;

class MyEventExample
{
    public static void Main()
    {
        try
        {

            // Creates a bitmask based on BindingFlags.
            BindingFlags myBindingFlags = BindingFlags.Instance |
BindingFlags.Public | BindingFlags.NonPublic;
            Type myTypeBindingFlags =
typeof(System.Windows.Forms.Button);
            EventInfo myEventBindingFlags =
myTypeBindingFlags.GetEvent("Click", myBindingFlags);
            if(myEventBindingFlags != null)
            {
                Console.WriteLine("Looking for the Click event in the
Button class with the specified BindingFlags.");
                Console.WriteLine(myEventBindingFlags.ToString());
            }
```

```
            else
            {
                Console.WriteLine("The Click event is not available with
the Button class.");
            }
        }
        catch(SecurityException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message :"+e.Message);
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message :"+e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("The following exception was raised :
{0}",e.Message);
        }
    }
}
```

Remarks

The following BindingFlags filter flags can be used to define which events to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public events in the search.
- Specify `BindingFlags.NonPublic` to include non-public events (that is, private, internal, and protected events) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the events declared on the Type, not events that were simply inherited.

See System.Reflection.BindingFlags for more information.

An event is considered public to reflection if it has at least one method or accessor that is public. Otherwise the event is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

If the current Type represents a constructed generic type, this method returns the EventInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the events of the class constraint.

See also

- EventInfo
- String
- BindingFlags
- DefaultBinder
- GetEvents()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetEvent(String)

Returns the EventInfo object representing the specified public event.

C#

```
public System.Reflection.EventInfo? GetEvent (string name);
```

Parameters

`name` String

The string containing the name of an event that is declared or inherited by the current Type.

Returns

EventInfo

The object representing the specified public event that is declared or inherited by the current Type, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following example creates an EventInfo object and gets the event for a button class for the specified event.

C#

```
using System;
using System.Reflection;
using System.Security;

class MyEventExample
{
    public static void Main()
    {
        try
        {

            Type myType = typeof(System.Windows.Forms.Button);
            EventInfo myEvent = myType.GetEvent("Click");
            if(myEvent != null)
            {
                Console.WriteLine("Looking for the Click event in the
Button class.");
                Console.WriteLine(myEvent.ToString());
            }
            else
            {
                Console.WriteLine("The Click event is not available in
the Button class.");
```

```
            }
        }
        catch(SecurityException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message :"+e.Message);
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message :"+e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("The following exception was raised :
{0}",e.Message);
        }
    }
}
```

Remarks

An event is considered public to reflection if it has at least one method or accessor that is public. Otherwise the event is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

The search for `name` is case-sensitive. The search includes public static and public instance events.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

Member Type	Static	Non-Static
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the EventInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the events of the class constraint.

See also

- EventInfo
- String
- DefaultBinder
- GetEvents()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetEvents Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the events that are declared or inherited by the current Type.

Overloads

Expand table

GetEvents()	Returns all the public events that are declared or inherited by the current Type.
GetEvents(BindingFlags)	When overridden in a derived class, searches for events that are declared or inherited by the current Type, using the specified binding constraints.

GetEvents()

Returns all the public events that are declared or inherited by the current Type.

C#

```
public virtual System.Reflection.EventInfo[] GetEvents ();
```

Returns

EventInfo[]

An array of EventInfo objects representing all the public events which are declared or inherited by the current Type.

-or-

An empty array of type EventInfo, if the current Type does not have public events.

Examples

The following example obtains an array of EventInfo objects, gets all the events for a `Button` class, and displays the event names. To compile the Visual Basic example, use the following command line:

```
vbc type_getevents1.vb /r:System.Windows.Forms.dll /r:System.dll
```

C#

```
using System;
using System.Reflection;
using System.Security;

class EventsSample
{
    public static void Main()
    {
        try
        {
            // Creates a bitmask based on BindingFlags.
            BindingFlags myBindingFlags = BindingFlags.Instance |
BindingFlags.Public;
            Type myTypeEvent = typeof(System.Windows.Forms.Button);
            EventInfo[] myEventsBindingFlags =
myTypeEvent.GetEvents(myBindingFlags);
            Console.WriteLine("\nThe events on the Button class with the
specified BindingFlags are : ");
            for (int index = 0; index < myEventsBindingFlags.Length;
index++)
            {

Console.WriteLine(myEventsBindingFlags[index].ToString());
            }
        }
        catch(SecurityException e)
        {
            Console.WriteLine("SecurityException :" + e.Message);
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException : " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception : " + e.Message);
        }
    }
}
```

Remarks

An event is considered public to reflection if it has at least one method or accessor that is public. Otherwise the event is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

In .NET 6 and earlier versions, the GetEvents method does not return events in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which events are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

This method can be overridden by a derived class.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the EventInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the events of the class constraint.

See also

- EventInfo
- GetEvent(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetEvents(BindingFlags)

When overridden in a derived class, searches for events that are declared or inherited by the current Type, using the specified binding constraints.

C#

```
public abstract System.Reflection.EventInfo[] GetEvents
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

EventInfo[]

An array of EventInfo objects representing all events that are declared or inherited by the current Type that match the specified binding constraints.

-or-

An empty array of type EventInfo, if the current Type does not have events, or if none of the events match the binding constraints.

Examples

The following example obtains an array of EventInfo objects that match the specified binding flags, gets all the events for a `Button` class, and displays the event names. To compile the Visual Basic example, use the following command line:

```
vbc type_getevents2.vb /r:System.Windows.Forms.dll /r:System.dll
```

C#

```
using System;
using System.Reflection;
using System.Security;

class EventsSample
{
    public static void Main()
    {
        try
        {
            // Create a bitmask based on BindingFlags.
            BindingFlags myBindingFlags = BindingFlags.Instance |
BindingFlags.Public;
            Type myTypeEvent = typeof(System.Windows.Forms.Button);
            EventInfo[] myEventsBindingFlags =
myTypeEvent.GetEvents(myBindingFlags);
```

```
            Console.WriteLine("\nThe events on the Button class with the
specified BindingFlags are:");
            for (int index = 0; index < myEventsBindingFlags.Length;
index++)
            {

Console.WriteLine(myEventsBindingFlags[index].ToString());
            }
        }
        catch(SecurityException e)
        {
            Console.WriteLine("SecurityException:" + e.Message);
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException: " + e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
    }
}
```

Remarks

In .NET 6 and earlier versions, the GetEvents method does not return events in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which events are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following BindingFlags filter flags can be used to define which events to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public events in the search.
- Specify `BindingFlags.NonPublic` to include non-public events (that is, private, internal, and protected events) in the search. Only protected and internal events on base classes are returned; private events on base classes are not returned.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not

included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the events declared on the Type, not events that were simply inherited.

See System.Reflection.BindingFlags for more information.

An event is considered public to reflection if it has at least one method or accessor that is public. Otherwise the event is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

If the current Type represents a constructed generic type, this method returns the EventInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the events of the class constraint.

See also

- EventInfo
- BindingFlags
- DefaultBinder
- GetEvent(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetField Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific field of the current Type.

Overloads

Expand table

GetField(String)	Searches for the public field with the specified name.
GetField(String, BindingFlags)	Searches for the specified field, using the specified binding constraints.

GetField(String)

Searches for the public field with the specified name.

C#

```
public System.Reflection.FieldInfo? GetField (string name);
```

Parameters

`name` String

The string containing the name of the data field to get.

Returns

FieldInfo

An object representing the public field with the specified name, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

NotSupportedException

This Type object is a TypeBuilder whose CreateType() method has not yet been called.

Examples

The following example gets the `Type` object for the specified class, obtains the FieldInfo object for the field, and displays the value of the field.

C#

```
using System;
using System.Reflection;

public class MyFieldClassA
{
    public string Field = "A Field";
}

public class MyFieldClassB
{
    private string field = "B Field";
    public string Field
    {
        get
        {
            return field;
        }
        set
        {
            if (field!=value)
            {
                field=value;
            }
        }
    }
}

public class MyFieldInfoClass
{
    public static void Main()
    {
        MyFieldClassB myFieldObjectB = new MyFieldClassB();
        MyFieldClassA myFieldObjectA = new MyFieldClassA();
```

```
        Type myTypeA = typeof(MyFieldClassA);
        FieldInfo myFieldInfo = myTypeA.GetField("Field");

        Type myTypeB = typeof(MyFieldClassB);
        FieldInfo myFieldInfo1 = myTypeB.GetField("field",
            BindingFlags.NonPublic | BindingFlags.Instance);

        Console.WriteLine("The value of the public field is: '{0}'",
            myFieldInfo.GetValue(myFieldObjectA));
        Console.WriteLine("The value of the private field is: '{0}'",
            myFieldInfo1.GetValue(myFieldObjectB));
    }
}
```

Remarks

The search for `name` is case-sensitive. The search includes public static and public instance fields.

If the current Type represents a constructed generic type, this method returns the FieldInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the fields of the class constraint.

See also

- FieldInfo
- String
- DefaultBinder
- GetFields()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetField(String, BindingFlags)

Searches for the specified field, using the specified binding constraints.

```
C#

public abstract System.Reflection.FieldInfo? GetField (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the data field to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

FieldInfo

An object representing the field that matches the specified requirements, if found; otherwise, `null`.

Implements

GetField(String, BindingFlags)

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following example gets the `Type` object for the specified class, obtains the FieldInfo object for the field that matches the specified binding flags, and displays

the value of the field.

C#

```
using System;
using System.Reflection;

public class MyFieldClassA
{
    public string Field = "A Field";
}

public class MyFieldClassB
{
    private string field = "B Field";
    public string Field
    {
        get
        {
            return field;
        }
        set
        {
            if (field!=value)
            {
                field=value;
            }
        }
    }
}

public class MyFieldInfoClass
{
    public static void Main()
    {
        MyFieldClassB myFieldObjectB = new MyFieldClassB();
        MyFieldClassA myFieldObjectA = new MyFieldClassA();

        Type myTypeA = typeof(MyFieldClassA);
        FieldInfo myFieldInfo = myTypeA.GetField("Field");

        Type myTypeB = typeof(MyFieldClassB);
        FieldInfo myFieldInfo1 = myTypeB.GetField("field",
            BindingFlags.NonPublic | BindingFlags.Instance);

        Console.WriteLine("The value of the public field is: '{0}'",
            myFieldInfo.GetValue(myFieldObjectA));
        Console.WriteLine("The value of the private field is: '{0}'",
            myFieldInfo1.GetValue(myFieldObjectB));
    }
}
```

Remarks

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

The following BindingFlags filter flags can be used to define which fields to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.

- Specify `BindingFlags.Public` to include public fields in the search.

- Specify `BindingFlags.NonPublic` to include non-public fields (that is, private, internal, and protected fields) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the fields declared on the Type, not fields that were simply inherited.

See System.Reflection.BindingFlags for more information.

If the current Type represents a constructed generic type, this method returns the FieldInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the fields of the class constraint.

See also

- FieldInfo
- String
- BindingFlags
- DefaultBinder
- GetFields()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetFields Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fields of the current Type.

Overloads

Expand table

GetFields()	Returns all the public fields of the current Type.
GetFields(BindingFlags)	When overridden in a derived class, searches for the fields defined for the current Type, using the specified binding constraints.

GetFields()

Returns all the public fields of the current Type.

C#

```
public System.Reflection.FieldInfo[] GetFields ();
```

Returns

FieldInfo[]

An array of FieldInfo objects representing all the public fields defined for the current Type.

-or-

An empty array of type FieldInfo, if no public fields are defined for the current Type.

Examples

The following example shows a use of the `GetFields()` method.

```C#
using System;
using System.Reflection;
using System.ComponentModel.Design;

class FieldInfo_IsSpecialName
{
    public static void Main()
    {
        try
        {
            // Get the type handle of a specified class.
            Type myType = typeof(ViewTechnology);

            // Get the fields of the specified class.
            FieldInfo[] myField = myType.GetFields();

            Console.WriteLine("\nDisplaying fields that have SpecialName
attributes:\n");
            for(int i = 0; i < myField.Length; i++)
            {
                // Determine whether or not each field is a special name.
                if(myField[i].IsSpecialName)
                {
                    Console.WriteLine("The field {0} has a SpecialName
attribute.",
                        myField[i].Name);
                }
            }
        }
        catch(Exception e)
        {
            Console.WriteLine("Exception : {0} " , e.Message);
        }
    }
}
```

Remarks

In .NET 6 and earlier versions, the GetFields method does not return fields in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which fields are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the FieldInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the public fields of the class constraint.

See also

- FieldInfo
- GetField(String, BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetFields(BindingFlags)

When overridden in a derived class, searches for the fields defined for the current Type, using the specified binding constraints.

C#

```
public abstract System.Reflection.FieldInfo[] GetFields
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

FieldInfo[]

An array of FieldInfo objects representing all fields defined for the current Type that match the specified binding constraints.

-or-

An empty array of type FieldInfo, if no fields are defined for the current Type, or if none of the defined fields match the binding constraints.

Implements

GetFields(BindingFlags)

Examples

The following example shows a use of the `GetFields(BindingFlags)` method.

C#

```
using System;
using System.Reflection;

class AttributesSample
{
    public void Mymethod (int int1m, out string str2m, ref string str3m)
    {
        str2m = "in Mymethod";
    }

    public static int Main(string[] args)
    {
        Console.WriteLine ("Reflection.MethodBase.Attributes Sample");

        // Get the type.
        Type MyType = Type.GetType("AttributesSample");

        // Get the method Mymethod on the type.
        MethodBase Mymethodbase = MyType.GetMethod("Mymethod");

        // Display the method name.
        Console.WriteLine("Mymethodbase = " + Mymethodbase);

        // Get the MethodAttribute enumerated value.
        MethodAttributes Myattributes = Mymethodbase.Attributes;

        // Display the flags that are set.
        PrintAttributes(typeof(System.Reflection.MethodAttributes), (int)
Myattributes);
        return 0;
    }

    public static void PrintAttributes(Type attribType, int iAttribValue)
    {
        if (!attribType.IsEnum)
        {
            Console.WriteLine("This type is not an enum.");
```

```
            return;
        }

        FieldInfo[] fields = attribType.GetFields(BindingFlags.Public |
BindingFlags.Static);
        for (int i = 0; i < fields.Length; i++)
        {
            int fieldvalue = (int)fields[i].GetValue(null);
            if ((fieldvalue & iAttribValue) == fieldvalue)
            {
                Console.WriteLine(fields[i].Name);
            }
        }
    }
}
```

Remarks

For the `GetFields(BindingFlags)` overload to successfully retrieve property information, the `bindingAttr` argument must include at least one of BindingFlags.Instance and BindingFlags.Static, along with at least one of BindingFlags.NonPublic and BindingFlags.Public.

The following BindingFlags filter flags can be used to define which fields to include in the search:

- Specify `BindingFlags.Instance` to include instance methods.
- Specify `BindingFlags.Static` to include static methods.
- Specify `BindingFlags.Public` to include public fields in the search.
- Specify `BindingFlags.NonPublic` to include non-public fields (that is, private, internal, and protected fields) in the search. Only protected and internal fields on base classes are returned; private fields on base classes are not returned.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.
- Specify `BindingFlags.Default` alone to return an empty PropertyInfo array.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the fields declared on the Type, not fields that were simply inherited.

See System.Reflection.BindingFlags for more information.

In .NET 6 and earlier versions, the GetFields method does not return fields in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which fields are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

If the current Type represents a constructed generic type, this method returns the FieldInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the public fields of the class constraint.

See also

- FieldInfo
- BindingFlags
- DefaultBinder
- GetField(String, BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.GetFunctionPointerCallingConventions Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the calling conventions of the current function pointer Type.

```
C#
public virtual Type[] GetFunctionPointerCallingConventions ();
```

Returns

Type[]

An array of Type objects representing all the calling conventions for the current function pointer Type.

-or-

An empty array of type Type, if no calling conventions are defined for the current function pointer Type.

-or-

An empty array of type Type, if the current function pointer Type is not a modified Type. A modified Type is obtained from GetModifiedFieldType(), GetModifiedPropertyType(), or GetModifiedParameterType().

Exceptions

InvalidOperationException

The current type is not a function pointer. That is, the IsFunctionPointer property returns `false`.

Remarks

This method is provided for designers of managed compilers. For more information on custom modifiers, see classes in the System.Runtime.CompilerServices namespace.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetFunctionPointerParameterTypes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the parameter types of the current function pointer Type.

```
C#

public virtual Type[] GetFunctionPointerParameterTypes ();
```

Returns

Type[]

An array of Type objects representing all the parameter types for the current function pointer Type.

-or-

An empty array of type Type, if no parameters are defined for the current function pointer Type.

Exceptions

InvalidOperationException

The current type is not a function pointer. That is, the IsFunctionPointer property returns `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetFunctionPointerReturnType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the return type of the current function pointer Type.

```
C#
public virtual Type GetFunctionPointerReturnType ();
```

Returns

Type

A Type object representing the return type for the current function pointer Type.

Exceptions

InvalidOperationException

The current type is not a function pointer. That is, the IsFunctionPointer property returns `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.GetGenericArguments Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an array of Type objects that represent the type arguments of a closed generic type or the type parameters of a generic type definition.

```
C#

public virtual Type[] GetGenericArguments ();
```

Returns

Type[]

An array of Type objects that represent the type arguments of a generic type. Returns an empty array if the current type is not a generic type.

Exceptions

NotSupportedException

The invoked method is not supported in the base class. Derived classes must provide an implementation.

Examples

The following code example uses the GetGenericArguments method to display the type arguments of a constructed type and the type parameters of its generic type definition.

This code example is part of a larger example provided for the IsGenericTypeDefinition property. See the larger example for sample output.

```
C#

if (t.IsGenericType)
{
    // If this is a generic type, display the type arguments.
    //
```

```
    Type[] typeArguments = t.GetGenericArguments();

    Console.WriteLine("\tList type arguments ({0}):",
        typeArguments.Length);

    foreach (Type tParam in typeArguments)
    {
        // If this is a type parameter, display its
        // position.
        //
        if (tParam.IsGenericParameter)
        {
            Console.WriteLine("\t\t{0}\t(unassigned - parameter position
{1})",
                tParam,
                tParam.GenericParameterPosition);
        }
        else
        {
            Console.WriteLine("\t\t{0}", tParam);
        }
    }
}
```

Remarks

The array elements are returned in the order in which they appear in the list of type arguments for the generic type.

- If the current type is a closed constructed type (that is, the ContainsGenericParameters property returns `false`), the array returned by the GetGenericArguments method contains the types that have been assigned to the generic type parameters of the generic type definition.
- If the current type is a generic type definition, the array contains the type parameters.
- If the current type is an open constructed type (that is, the ContainsGenericParameters property returns `true`) in which specific types have not been assigned to all of the type parameters and type parameters of enclosing generic types or methods, the array contains both types and type parameters. Use the IsGenericParameter property to tell them apart. For a demonstration of this scenario, see the code example for the ContainsGenericParameters property.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ContainsGenericParameters
- IsGenericTypeDefinition
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetGenericParameterConstraints Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an array of Type objects that represent the constraints on the current generic type parameter.

C#

```
public virtual Type[] GetGenericParameterConstraints ();
```

Returns

Type[]

An array of Type objects that represent the constraints on the current generic type parameter.

Exceptions

InvalidOperationException

The current Type object is not a generic type parameter. That is, the IsGenericParameter property returns `false`.

Examples

The following code example defines a generic type `Test` with two type parameters that have different constraints. When the program executes, the constraints are examined using the GenericParameterAttributes property and the GetGenericParameterConstraints method.

C#

```
using System;
using System.Reflection;
```

```
// Define a sample interface to use as an interface constraint.
public interface ITest {}

// Define a base type to use as a base class constraint.
public class Base {}

// Define the generic type to examine. The first generic type parameter,
// T, derives from the class Base and implements ITest. This demonstrates
// a base class constraint and an interface constraint. The second generic
// type parameter, U, must be a reference type (class) and must have a
// default constructor (new()). This demonstrates special constraints.
//
public class Test<T,U>
    where T : Base, ITest
    where U : class, new() {}

// Define a type that derives from Base and implements ITest. This type
// satisfies the constraints on T in class Test.
public class Derived : Base, ITest {}

public class Example
{
    public static void Main()
    {
        // To get the generic type definition, omit the type
        // arguments but retain the comma to indicate the number
        // of type arguments.
        //
        Type def = typeof(Test<,>);
        Console.WriteLine("\r\nExamining generic type {0}", def);

        // Get the type parameters of the generic type definition,
        // and display them.
        //
        Type[] defparams = def.GetGenericArguments();
        foreach (Type tp in defparams)
        {
            Console.WriteLine("\r\nType parameter: {0}", tp.Name);
            Console.WriteLine("\t{0}",
                ListGenericParameterAttributes(tp));

            // List the base class and interface constraints. The
            // constraints are returned in no particular order. If
            // there are no class or interface constraints, an empty
            // array is returned.
            //
            Type[] tpConstraints = tp.GetGenericParameterConstraints();
            foreach (Type tpc in tpConstraints)
            {
                Console.WriteLine("\t{0}", tpc);
            }
        }
    }
```

```
    // List the variance and special constraint flags.
    //
    private static string ListGenericParameterAttributes(Type t)
    {
        string retval;
        GenericParameterAttributes gpa = t.GenericParameterAttributes;
        GenericParameterAttributes variance = gpa &
            GenericParameterAttributes.VarianceMask;

        // Select the variance flags.
        if (variance == GenericParameterAttributes.None)
        {
            retval = "No variance flag;";
        }
        else
        {
            if ((variance & GenericParameterAttributes.Covariant) != 0)
                retval = "Covariant;";
            else
                retval = "Contravariant;";
        }

        // Select
        GenericParameterAttributes constraints = gpa &
            GenericParameterAttributes.SpecialConstraintMask;

        if (constraints == GenericParameterAttributes.None)
        {
            retval += " No special constraints";
        }
        else
        {
            if ((constraints &
GenericParameterAttributes.ReferenceTypeConstraint) != 0)
                retval += " ReferenceTypeConstraint";
            if ((constraints &
GenericParameterAttributes.NotNullableValueTypeConstraint) != 0)
                retval += " NotNullableValueTypeConstraint";
            if ((constraints &
GenericParameterAttributes.DefaultConstructorConstraint) != 0)
                retval += " DefaultConstructorConstraint";
        }

        return retval;
    }
}
/* This example produces the following output:

Examining generic type Test`2[T,U]

Type parameter: T
        No variance flag; no special constraints.
        Base
        ITest
```

```
Type parameter: U
        No variance flag; ReferenceTypeConstraint
DefaultConstructorConstraint
 */
```

Remarks

Each constraint on a generic type parameter is expressed as a Type object. Use the IsClass property to determine whether a constraint is the base class constraint; if the property returns `false`, the constraint is an interface constraint. If a type parameter has no class constraint and no interface constraints, an empty array is returned.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GenericParameterAttributes
- IsGenericParameter
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetGenericTypeDefinition Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a Type object that represents a generic type definition from which the current generic type can be constructed.

```
C#
public virtual Type GetGenericTypeDefinition ();
```

Returns

Type

A Type object representing a generic type from which the current type can be constructed.

Exceptions

InvalidOperationException

The current type is not a generic type. That is, IsGenericType returns `false`.

NotSupportedException

The invoked method is not supported in the base class. Derived classes must provide an implementation.

Examples

The following code example creates an instance of a constructed type by using ordinary instance creation and then uses the GetType and GetGenericTypeDefinition methods to retrieve the constructed type and the generic type definition. This example uses the generic Dictionary<TKey,TValue> type; the constructed type represents a Dictionary<TKey,TValue> of `Test` objects with string keys.

```
C#
```

```
using System;
using System.Reflection;
using System.Collections.Generic;

public class Test
{
    public static void Main()
    {
        Console.WriteLine("\r\n--- Get the generic type that defines a
constructed type.");

        // Create a Dictionary of Test objects, using strings for the
        // keys.
        Dictionary<string, Test> d = new Dictionary<string, Test>();

        // Get a Type object representing the constructed type.
        //
        Type constructed = d.GetType();
        DisplayTypeInfo(constructed);

        Type generic = constructed.GetGenericTypeDefinition();
        DisplayTypeInfo(generic);
    }

    private static void DisplayTypeInfo(Type t)
    {
        Console.WriteLine("\r\n{0}", t);
        Console.WriteLine("\tIs this a generic type definition? {0}",
            t.IsGenericTypeDefinition);
        Console.WriteLine("\tIs it a generic type? {0}",
            t.IsGenericType);
        Type[] typeArguments = t.GetGenericArguments();
        Console.WriteLine("\tList type arguments ({0}):",
typeArguments.Length);
        foreach (Type tParam in typeArguments)
        {
            Console.WriteLine("\t\t{0}", tParam);
        }
    }
}

/* This example produces the following output:

--- Get the generic type that defines a constructed type.

System.Collections.Generic.Dictionary`2[System.String,Test]
        Is this a generic type definition? False
        Is it a generic type? True
        List type arguments (2):
                System.String
                Test

System.Collections.Generic.Dictionary`2[TKey,TValue]
        Is this a generic type definition? True
```

```
        Is it a generic type? True
        List type arguments (2):
                TKey
                TValue
 */
```

Remarks

A generic type definition is a template from which other types can be constructed. For example, from the generic type definition `G<T>` (expressed in C# syntax; `G(Of T)` in Visual Basic or `generic <typename T> ref class G` in C++) you can construct and instantiate the type `G<int>` (`G(Of Integer)` in Visual Basic). Given a Type object representing this constructed type, the GetGenericTypeDefinition method returns the generic type definition.

If two constructed types are created from the same generic type definition, using the same type arguments, the GetGenericTypeDefinition method returns the same Type object for both types.

If you call the GetGenericTypeDefinition method on a Type object that already represents a generic type definition, it returns the current Type.

> **Important**
>
> An array of generic types is not itself generic. In the C# code `A<int>[] v;` or the Visual Basic code `Dim v() As A(Of Integer)`, the type of variable `v` is not generic. Use **IsGenericType** to determine whether a type is generic before calling **GetGenericTypeDefinition**.

For a list of the invariant conditions for terms used in generic reflection, see the IsGenericType property remarks.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- MakeGenericType(Type[])
- ContainsGenericParameters
- IsGenericTypeDefinition
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for this instance.

Examples

The following example displays the hash code of the `System.Windows.Forms.Button` class.

C#

```
using System;
using System.Security;
using System.Reflection;

class FieldsSample
{
    public static void Main()
    {
        Type myType = typeof(System.Net.IPAddress);
        FieldInfo [] myFields = myType.GetFields(BindingFlags.Static |
BindingFlags.NonPublic);
        Console.WriteLine ("\nThe IPAddress class has the following
nonpublic fields: ");
        foreach (FieldInfo myField in myFields)
        {
            Console.WriteLine(myField.ToString());
        }
        Type myType1 = typeof(System.Net.IPAddress);
        FieldInfo [] myFields1 = myType1.GetFields();
```

```
        Console.WriteLine ("\nThe IPAddress class has the following public
fields: ");
        foreach (FieldInfo myField in myFields1)
        {
            Console.WriteLine(myField.ToString());
        }
        try
        {
            Console.WriteLine("The HashCode of the
System.Windows.Forms.Button type is: {0}",
                typeof(System.Windows.Forms.Button).GetHashCode());
        }
        catch(SecurityException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message: "+e.Message);
        }
        catch(Exception e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Message: "+e.Message);
        }
    }
}
```

Remarks

This method overrides Object.GetHashCode.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.GetInterface Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific interface implemented or inherited by the current Type.

Overloads

Expand table

GetInterface(String)	Searches for the interface with the specified name.
GetInterface(String, Boolean)	When overridden in a derived class, searches for the specified interface, specifying whether to do a case-insensitive search for the interface name.

GetInterface(String)

Searches for the interface with the specified name.

C#

```
public Type? GetInterface (string name);
```

Parameters

`name` String

The string containing the name of the interface to get. For generic interfaces, this is the mangled name.

Returns

Type

An object representing the interface with the specified name, implemented or inherited by the current Type, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

AmbiguousMatchException

The current Type represents a type that implements the same generic interface with different type arguments.

Examples

The following code example uses the GetInterface(String) method to search the Hashtable class for the IDeserializationCallback interface, and lists the methods of the interface.

The code example also demonstrates the GetInterface(String, Boolean) method overload and the GetInterfaceMap method.

C#

```
public static void Main()
{
    Hashtable hashtableObj = new Hashtable();
    Type objType = hashtableObj.GetType();
    MethodInfo[] arrayMethodInfo;
    MemberInfo[] arrayMemberInfo;
    try
    {
        // Get the methods implemented in 'IDeserializationCallback'
interface.
        arrayMethodInfo
=objType.GetInterface("IDeserializationCallback").GetMethods();
        Console.WriteLine ("\nMethods of 'IDeserializationCallback'
Interface :");
        foreach(MethodInfo methodInfo in arrayMethodInfo)
            Console.WriteLine (methodInfo);

        // Get FullName for interface by using Ignore case search.
        Console.WriteLine ("\nMethods of 'IEnumerable' Interface");
        arrayMethodInfo =
objType.GetInterface("ienumerable",true).GetMethods();
        foreach(MethodInfo methodInfo in arrayMethodInfo)
           Console.WriteLine (methodInfo);

        //Get the Interface methods for 'IDictionary' interface
```

```
        InterfaceMapping interfaceMappingOb =
objType.GetInterfaceMap(typeof(IDictionary));
        arrayMemberInfo = interfaceMappingObj.InterfaceMethods;
        Console.WriteLine ("\nHashtable class Implements the following
IDictionary Interface methods :");
        foreach(MemberInfo memberInfo in arrayMemberInfo)
           Console.WriteLine (memberInfo);
    }
    catch (Exception e)
    {
        Console.WriteLine ("Exception : " + e.ToString());
    }
}
```

Remarks

The search for `name` is case-sensitive.

If the current Type represents a constructed generic type, this method returns the Type with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the interface constraints and any interfaces inherited from class or interface constraints.

> **Note**
>
> For generic interfaces, the `name` parameter is the mangled name, ending with a grave accent (`) and the number of type parameters. This is true for both generic interface definitions and constructed generic interfaces. For example, to find `IExample<T>` (`IExample(Of T)` in Visual Basic) or `IExample<string>` (`IExample(Of String)` in Visual Basic), search for `"IExample`1"`.

See also

- String
- GetInterfaces()
- FindInterfaces(TypeFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetInterface(String, Boolean)

When overridden in a derived class, searches for the specified interface, specifying whether to do a case-insensitive search for the interface name.

C#

```
public abstract Type? GetInterface (string name, bool ignoreCase);
```

Parameters

`name` String

The string containing the name of the interface to get. For generic interfaces, this is the mangled name.

`ignoreCase` Boolean

`true` to ignore the case of that part of `name` that specifies the simple interface name (the part that specifies the namespace must be correctly cased).

-or-

`false` to perform a case-sensitive search for all parts of `name`.

Returns

Type

An object representing the interface with the specified name, implemented or inherited by the current Type, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

AmbiguousMatchException

The current Type represents a type that implements the same generic interface with different type arguments.

Examples

The following code example uses the GetInterface(String, Boolean) method to perform a case-insensitive search of the Hashtable class for the IEnumerable interface.

The code example also demonstrates the GetInterface(String) method overload and the GetInterfaceMap method.

C#

```
public static void Main()
{
    Hashtable hashtableObj = new Hashtable();
    Type objType = hashtableObj.GetType();
    MethodInfo[] arrayMethodInfo;
    MemberInfo[] arrayMemberInfo;
    try
    {
        // Get the methods implemented in 'IDeserializationCallback'
interface.
        arrayMethodInfo
=objType.GetInterface("IDeserializationCallback").GetMethods();
        Console.WriteLine ("\nMethods of 'IDeserializationCallback'
Interface :");
        foreach(MethodInfo methodInfo in arrayMethodInfo)
            Console.WriteLine (methodInfo);

        // Get FullName for interface by using Ignore case search.
        Console.WriteLine ("\nMethods of 'IEnumerable' Interface");
        arrayMethodInfo =
objType.GetInterface("ienumerable",true).GetMethods();
        foreach(MethodInfo methodInfo in arrayMethodInfo)
           Console.WriteLine (methodInfo);

        //Get the Interface methods for 'IDictionary' interface
        InterfaceMapping interfaceMappingOb =
objType.GetInterfaceMap(typeof(IDictionary));
        arrayMemberInfo = interfaceMappingObj.InterfaceMethods;
        Console.WriteLine ("\nHashtable class Implements the following
IDictionary Interface methods :");
        foreach(MemberInfo memberInfo in arrayMemberInfo)
           Console.WriteLine (memberInfo);
    }
    catch (Exception e)
    {
```

```
        Console.WriteLine ("Exception : " + e.ToString());
    }
}
```

Remarks

The `ignoreCase` parameter applies only to the simple interface name, not to the namespace. The portion of `name` that specifies the namespace must have the correct case, or the interface will not be found. For example, the string "System.icomparable" finds the IComparable interface, but the string "system.icomparable" does not.

If the current Type represents a constructed generic type, this method returns the Type with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the interface constraints and any interfaces inherited from class or interface constraints.

> **Note**
>
> For generic interfaces, the `name` parameter is the mangled name, ending with a grave accent (`) and the number of type parameters. This is true for both generic interface definitions and constructed generic interfaces. For example, to find `IExample<T>` (`IExample(Of T)` in Visual Basic) or `IExample<string>` (`IExample(Of String)` in Visual Basic), search for `"IExample`1"`.

See also

- String
- GetInterfaces()
- FindInterfaces(TypeFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetInterfaceMap(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an interface mapping for the specified interface type.

C#

```
public virtual System.Reflection.InterfaceMapping GetInterfaceMap (Type
interfaceType);
```

Parameters

`interfaceType` Type

The interface type to retrieve a mapping for.

Returns

InterfaceMapping

An object that represents the interface mapping for `interfaceType`.

Exceptions

ArgumentException

`interfaceType` is not implemented by the current type.

-or-

The `interfaceType` argument does not refer to an interface.

-or-

The current instance or `interfaceType` argument is an open generic type; that is, the ContainsGenericParameters property returns `true`.

-or-

`interfaceType` is a generic interface, and the current type is an array type.

ArgumentNullException

`interfaceType` is `null`.

InvalidOperationException

The current Type represents a generic type parameter; that is, IsGenericParameter is `true`.

NotSupportedException

The invoked method is not supported in the base class. Derived classes must provide an implementation.

Examples

The following example calls the GetInterfaceMap method to determine how the IFormatProvider interface maps to CultureInfo methods, and how the IAppDomainSetup interface maps to AppDomainSetup properties. Note that, because the IAppDomainSetup interface defines a set of properties, the returned InterfaceMapping object includes separate MethodInfo objects for a property's get and set accessors.

C#

```
using System;
using System.Globalization;
using System.Reflection;

public class Example
{
   public static void Main()
   {
      Type[] interf = { typeof(IFormatProvider), typeof(IAppDomainSetup) };
      Type[] impl = { typeof(CultureInfo), typeof(AppDomainSetup) };

      for (int ctr = 0; ctr < interf.Length; ctr++)
         ShowInterfaceMapping(interf[ctr], impl[ctr]);
   }

   private static void ShowInterfaceMapping(Type intType, Type implType)
   {
      InterfaceMapping map = implType.GetInterfaceMap(intType);
      Console.WriteLine("Mapping of {0} to {1}: ", map.InterfaceType,
map.TargetType);
      for (int ctr = 0; ctr < map.InterfaceMethods.Length; ctr++) {
         MethodInfo im = map.InterfaceMethods[ctr];
         MethodInfo tm = map.TargetMethods[ctr];
         Console.WriteLine("   {0} --> {1}", im.Name,tm.Name);
```

```
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//    Mapping of System.IFormatProvider to System.Globalization.CultureInfo:
//       GetFormat --> GetFormat
//
//    Mapping of System.IAppDomainSetup to System.AppDomainSetup:
//       get_ApplicationBase --> get_ApplicationBase
//       set_ApplicationBase --> set_ApplicationBase
//       get_ApplicationName --> get_ApplicationName
//       set_ApplicationName --> set_ApplicationName
//       get_CachePath --> get_CachePath
//       set_CachePath --> set_CachePath
//       get_ConfigurationFile --> get_ConfigurationFile
//       set_ConfigurationFile --> set_ConfigurationFile
//       get_DynamicBase --> get_DynamicBase
//       set_DynamicBase --> set_DynamicBase
//       get_LicenseFile --> get_LicenseFile
//       set_LicenseFile --> set_LicenseFile
//       get_PrivateBinPath --> get_PrivateBinPath
//       set_PrivateBinPath --> set_PrivateBinPath
//       get_PrivateBinPathProbe --> get_PrivateBinPathProbe
//       set_PrivateBinPathProbe --> set_PrivateBinPathProbe
//       get_ShadowCopyDirectories --> get_ShadowCopyDirectories
//       set_ShadowCopyDirectories --> set_ShadowCopyDirectories
//       get_ShadowCopyFiles --> get_ShadowCopyFiles
//       set_ShadowCopyFiles --> set_ShadowCopyFiles
```

Remarks

The interface map denotes how an interface is mapped into the actual members on a class that implements that interface.

If the current Type represents a constructed generic type, type parameters are replaced by the appropriate type arguments in the elements of the InterfaceMapping returned by this method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- InterfaceMapping

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetInterfaces Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, gets all the interfaces implemented or inherited by the current Type.

C#

```
public abstract Type[] GetInterfaces ();
```

Returns

Type[]

An array of Type objects representing all the interfaces implemented or inherited by the current Type.

-or-

An empty array of type Type, if no interfaces are implemented or inherited by the current Type.

Exceptions

TargetInvocationException

A static initializer is invoked and throws an exception.

Examples

The following example gets the type of the specified class and displays all the interfaces that the type implements or inherits. To compile the Visual Basic example, use the following compiler commands:

```
vbc type_getinterfaces1.vb /r:System.Web.dll /r:System.dll
```

C#

```
using System;
using System.Collections.Generic;

public class Example
{
    static void Main()
    {
        Console.WriteLine("\r\nInterfaces implemented by Dictionary<int,
string>:\r\n");

        foreach (Type tinterface in typeof(Dictionary<int,
string>).GetInterfaces())
        {
            Console.WriteLine(tinterface.ToString());
        }

        //Console.ReadLine()      // Uncomment this line for Visual Studio.
    }
}

/* This example produces output similar to the following:

Interfaces implemented by Dictionary<int, string>:

System.Collections.Generic.IDictionary`2[System.Int32,System.String]
System.Collections.Generic.ICollection`1[System.Collections.Generic.KeyValue
Pair`2[System.Int32,System.String]]
System.Collections.Generic.IEnumerable`1[System.Collections.Generic.KeyValue
Pair`2[System.Int32,System.String]]
System.Collection.IEnumerable
System.Collection.IDictionary
System.Collection.ICollection
System.Runtime.Serialization.ISerializable
System.Runtime.Serialization.IDeserializationCallback
 */
```

Remarks

In .NET 6 and earlier versions, the GetInterfaces method does not return interfaces in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which interfaces are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

If the current Type represents a constructed generic type, this method returns the Type objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the interface constraints and any interfaces

inherited from class or interface constraints.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- GetInterface(String)
- FindInterfaces(TypeFilter, Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetMember Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the specified members of the current Type.

Overloads

Expand table

GetMember(String)	Searches for the public members with the specified name.
GetMember(String, BindingFlags)	Searches for the specified members, using the specified binding constraints.
GetMember(String, MemberTypes, BindingFlags)	Searches for the specified members of the specified member type, using the specified binding constraints.

GetMember(String)

Searches for the public members with the specified name.

C#

```
public System.Reflection.MemberInfo[] GetMember (string name);
```

Parameters

name String

The string containing the name of the public members to get.

Returns

MemberInfo[]

An array of MemberInfo objects representing the public members with the specified name, if found; otherwise, an empty array.

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following example displays all the members of the `String` class that start with the letter C.

C#

```
using System;
using System.Security;
using System.Reflection;

public class MyMemberSample
{
    public static void Main()
    {
        MyMemberSample myClass = new MyMemberSample();
        try
        {
            myClass.GetMemberInfo();
            myClass.GetPublicStaticMemberInfo();
            myClass.GetPublicInstanceMethodMemberInfo();
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException occurred.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
        catch(NotSupportedException e)
        {
            Console.WriteLine("NotSupportedException occurred.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
        catch(SecurityException e)
        {
            Console.WriteLine("SecurityException occurred.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
        catch(Exception e)
```

```
        {
            Console.WriteLine("Exception occurred.");
            Console.WriteLine("Source: " + e.Source);
            Console.WriteLine("Message: " + e.Message);
        }
    }

    public void GetMemberInfo()
    {
        String myString = "GetMember_String";

        Type myType = myString.GetType();
        // Get the members for myString starting with the letter C.
        MemberInfo[] myMembers = myType.GetMember("C*");
        if(myMembers.Length > 0)
        {
            Console.WriteLine("\nThe member(s) starting with the letter C
for type {0}:", myType);
            for(int index=0; index < myMembers.Length; index++)
                Console.WriteLine("Member {0}: {1}", index + 1,
myMembers[index].ToString());
        }
        else
        {
            Console.WriteLine("No members match the search criteria.");
        }
    }
```

Remarks

The search for `name` is case-sensitive. The search includes public static and public instance members.

Members include properties, methods, fields, events, and so on.

In .NET 6 and earlier versions, the GetMember method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

This method overload will not find class initializers (static constructor). To find class initializers, use an overload that takes BindingFlags, and specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the MemberInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text

name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MemberInfo
- String
- DefaultBinder
- GetMembers()
- GetDefaultMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMember(String, BindingFlags)

Searches for the specified members, using the specified binding constraints.

C#

```
public virtual System.Reflection.MemberInfo[] GetMember (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the members to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

MemberInfo[]

An array of MemberInfo objects representing the public members with the specified name, if found; otherwise, an empty array.

Implements

GetMember(String, BindingFlags)

Exceptions

ArgumentNullException

`name` is `null`.

Examples

The following example displays all the public static members of the `myString` class that start with the letter C.

C#

```
public void GetPublicStaticMemberInfo()
{
    String myString = "GetMember_String_BindingFlag";
    Type myType = myString.GetType();
    // Get the public static members for the class myString starting with
the letter C.
    MemberInfo[] myMembers = myType.GetMember("C*",
        BindingFlags.Public |BindingFlags.Static);
    if(myMembers.Length > 0)
    {
        Console.WriteLine("\nThe public static member(s) starting with
the letter C for type {0}:", myType);
        for(int index=0; index < myMembers.Length; index++)
            Console.WriteLine("Member {0}: {1}", index + 1,
myMembers[index].ToString());
    }
    else
```

```
    {
        Console.WriteLine("No members match the search criteria.");
    }
}
```

Remarks

This method can be overridden by a derived class.

Members include properties, methods, fields, events, and so on.

In .NET 6 and earlier versions, the GetMember method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following BindingFlags filter flags can be used to define which members to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public members in the search.
- Specify `BindingFlags.NonPublic` to include non-public members (that is, private, internal, and protected members) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

See System.Reflection.BindingFlags for more information.

To get the class initializer (static constructor) using this method overload, you must specify "static constructor" for `name`, and BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual

Basic) for `bindingAttr`. You can also get the class initializer using the TypeInitializer property.

If the current Type represents a constructed generic type, this method returns the MemberInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MemberInfo
- String
- BindingFlags
- DefaultBinder
- GetMembers()
- GetDefaultMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMember(String, MemberTypes, BindingFlags)

Searches for the specified members of the specified member type, using the specified binding constraints.

C#

```
public virtual System.Reflection.MemberInfo[] GetMember (string name, System.Reflection.MemberTypes type, System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the members to get.

`type` MemberTypes

The value to search for.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

MemberInfo[]

An array of MemberInfo objects representing the public members with the specified name, if found; otherwise, an empty array.

Exceptions

ArgumentNullException

`name` is `null`.

NotSupportedException

A derived class must provide an implementation.

Examples

The following example displays all the methods of the `myString` class that start with the letter C.

C#

```
    public void GetPublicInstanceMethodMemberInfo()
    {
        String myString = "GetMember_String_MemberType_BindingFlag";
        Type myType = myString.GetType();
        // Get the public instance methods for myString starting with the
letter C.
        MemberInfo[] myMembers = myType.GetMember("C*",
MemberTypes.Method,
            BindingFlags.Public | BindingFlags.Instance);
        if(myMembers.Length > 0)
        {
            Console.WriteLine("\nThe public instance method(s) starting
with the letter C for type {0}:", myType);
            for(int index=0; index < myMembers.Length; index++)
                Console.WriteLine("Member {0}: {1}", index + 1,
myMembers[index].ToString());
        }
        else
        {
            Console.WriteLine("No members match the search criteria.");
        }
    }
}
```

Remarks

Members include properties, methods, fields, events, and so on.

In .NET 6 and earlier versions, the GetMember method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following BindingFlags filter flags can be used to define which members to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.

- Specify `BindingFlags.Public` to include public members in the search.
- Specify `BindingFlags.NonPublic` to include non-public members (that is, private, internal, and protected members) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

See System.Reflection.BindingFlags for more information.

To get the class initializer (static constructor) using this method overload, you must specify "static constructor" for `name`, MemberTypes.Constructor for `type`, and BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic) for `bindingAttr`. You can also get the class initializer using the TypeInitializer property.

If the current Type represents a constructed generic type, this method returns the MemberInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MemberInfo

- String
- BindingFlags
- DefaultBinder
- GetMembers()
- GetDefaultMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetMembers Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the members (properties, methods, fields, events, and so on) of the current Type.

Overloads

Expand table

GetMembers(BindingFlags)	When overridden in a derived class, searches for the members defined for the current Type, using the specified binding constraints.
GetMembers()	Returns all the public members of the current Type.

GetMembers(BindingFlags)

When overridden in a derived class, searches for the members defined for the current Type, using the specified binding constraints.

C#

```
public abstract System.Reflection.MemberInfo[] GetMembers
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

bindingAttr BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

MemberInfo[]

An array of MemberInfo objects representing all members defined for the current Type that match the specified binding constraints.

-or-

An empty array if no members are defined for the current Type, or if none of the defined members match the binding constraints.

Implements

GetMembers(BindingFlags)

Examples

The following code example demonstrates how to use the GetMembers(BindingFlags) method overload to collect information about all public instance members of a specified class.

C#

```
class MyClass
{
   public int myInt = 0;
   public string myString = null;

   public MyClass()
   {
   }
   public void Myfunction()
   {
   }
}

class Type_GetMembers_BindingFlags
{
   public static void Main()
   {
      try
      {
         MyClass MyObject = new MyClass();
         MemberInfo [] myMemberInfo;

         // Get the type of the class 'MyClass'.
         Type myType = MyObject.GetType();
```

```
         // Get the public instance members of the class 'MyClass'.
         myMemberInfo =
myType.GetMembers(BindingFlags.Public|BindingFlags.Instance);

         Console.WriteLine( "\nThe public instance members of class '{0}'
are : \n", myType);
         for (int i =0 ; i < myMemberInfo.Length ; i++)
         {
            // Display name and type of the member of 'MyClass'.
            Console.WriteLine( "'{0}' is a {1}", myMemberInfo[i].Name,
myMemberInfo[i].MemberType);
         }
      }
      catch (SecurityException e)
      {
         Console.WriteLine("SecurityException : " + e.Message );
      }

      //Output:
      //The public instance members of class 'MyClass' are :

      //'Myfunction' is a Method
      //'ToString' is a Method
      //'Equals' is a Method
      //'GetHashCode' is a Method
      //'GetType' is a Method
      //'.ctor' is a Constructor
      //'myInt' is a Field
      //'myString' is a Field
   }
}
```

Remarks

Members include properties, methods, constructors, fields, events, and nested types.

For the `GetMethods(BindingFlags)` overload to successfully retrieve method information, the `bindingAttr` argument must include at least one of BindingFlags.Instance and BindingFlags.Static, along with at least one of BindingFlags.NonPublic and BindingFlags.Public. The sole exception is a method call with BindingFlags.NonPublic, which returns member information about nested types.

The following BindingFlags filter flags can be used to define which members to include in the search:

- Specify `BindingFlags.Instance` to include instance methods.
- Specify `BindingFlags.Static` to include static methods.

- Specify `BindingFlags.Public` to include public methods in the search.
- Specify `BindingFlags.NonPublic` to include non-public methods (that is, private, internal, and protected methods) in the search. Only protected and internal methods on base classes are returned; private methods on base classes are not returned.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.
- Specify `BindingFlags.Default` alone to return an empty MethodInfo array.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

See System.Reflection.BindingFlags for more information.

In .NET 6 and earlier versions, the GetMembers method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

To get the class initializer (static constructor) using this method overload, you must specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

If the current Type represents a constructed generic type, this method returns the MemberInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

See also

- MemberInfo
- BindingFlags

- DefaultBinder
- GetMember(String)
- GetDefaultMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMembers()

Returns all the public members of the current Type.

C#

```
public System.Reflection.MemberInfo[] GetMembers ();
```

Returns

MemberInfo[]

An array of MemberInfo objects representing all the public members of the current Type.

-or-

An empty array of type MemberInfo, if the current Type does not have public members.

Examples

The following code example demonstrates how to use the GetMembers() method overload to collect information about all public members of a specified class.

C#

```
class MyClass
{
   public int myInt = 0;
   public string myString = null;

   public MyClass()
   {
   }
   public void Myfunction()
   {
   }
}

class Type_GetMembers
{
   public static void Main()
   {
      try
      {
         MyClass myObject = new MyClass();
         MemberInfo[] myMemberInfo;

         // Get the type of 'MyClass'.
         Type myType = myObject.GetType();

         // Get the information related to all public member's of
'MyClass'.
         myMemberInfo = myType.GetMembers();

         Console.WriteLine( "\nThe members of class '{0}' are :\n",
myType);
         for (int i =0 ; i < myMemberInfo.Length ; i++)
         {
            // Display name and type of the concerned member.
            Console.WriteLine( "'{0}' is a {1}", myMemberInfo[i].Name,
myMemberInfo[i].MemberType);
         }
      }
      catch(SecurityException e)
      {
         Console.WriteLine("Exception : " + e.Message );
      }
   }
}
```

Remarks

Members include properties, methods, constructors, fields, events, and nested types.

In .NET 6 and earlier versions, the GetMembers method does not return members in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which members are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

This method overload calls the GetMembers(BindingFlags) method overload, with BindingFlags.Public | BindingFlags.Instance | BindingFlags.Static (BindingFlags.Public `Or` BindingFlags.Instance `Or` BindingFlags.Static in Visual Basic). It will not find class initializers (static constructors). To find class initializers, call the GetMembers(BindingFlags) overload, and specify BindingFlags.Static | BindingFlags.NonPublic (BindingFlags.Static `Or` BindingFlags.NonPublic in Visual Basic). You can also get the class initializer using the TypeInitializer property.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the MemberInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the members of the class constraint, or the members of Object if there is no class constraint.

See also

- MemberInfo
- GetMember(String)
- GetDefaultMembers()
- FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Type.GetMemberWithSameMetadataDefinitionAs(MemberInfo) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Searches for the MemberInfo on the current Type that matches the specified MemberInfo.

C#

```
public virtual System.Reflection.MemberInfo
GetMemberWithSameMetadataDefinitionAs (System.Reflection.MemberInfo member);
```

Parameters

member MemberInfo

The MemberInfo to find on the current Type.

Returns

MemberInfo

An object representing the member on the current Type that matches the specified member.

Exceptions

ArgumentNullException

`member` is `null`.

ArgumentException

`member` does not match a member on the current Type.

Remarks

This method can be used to find a constructed generic member given a member from a generic type definition.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetMethod Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific method of the current Type.

Overloads

Expand table

GetMethod(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethod(String, Int32, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints.
GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetMethod(String, Int32, Type[], ParameterModifier[])	Searches for the specified public method whose parameters match the specified generic parameter count, argument types and modifiers.
GetMethod(String, BindingFlags)	Searches for the specified method, using the specified binding constraints.
GetMethod(String, BindingFlags, Type[])	Searches for the specified method whose parameters match the specified argument types, using the specified binding constraints.

GetMethod(String, Int32, Type[])	Searches for the specified public method whose parameters match the specified generic parameter count and argument types.
GetMethod(String, Type[])	Searches for the specified public method whose parameters match the specified argument types.
GetMethod(String)	Searches for the public method with the specified name.
GetMethod(String, Type[], ParameterModifier[])	Searches for the specified public method whose parameters match the specified argument types and modifiers.

GetMethod(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, int
genericParameterCount, System.Reflection.BindingFlags bindingAttr,
System.Reflection.Binder? binder, System.Reflection.CallingConventions
callConvention, Type[] types, System.Reflection.ParameterModifier[]?
modifiers);
```

Parameters

name String

The string containing the name of the public method to get.

genericParameterCount Int32

The number of generic type parameters of the method.

bindingAttr BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and how the stack is cleaned up.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified generic parameter count, argument types, modifiers, binding constraints and calling convention, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in the `types` array is `null`.

ArgumentException

`genericParameterCount` is negative.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, System.Reflection.CallingConventions callConvention, Type[]
types, System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

name String

The string containing the name of the method to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and how the stack is cleaned up.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified requirements, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

Examples

The following example finds specific overloads of `MethodA`, specifying binding constraints, calling conventions, and a variety of argument types.

> **Note**
>
> The Visual C# 2005 example requires the `/unsafe` compiler option.

C#

```
using System;
using System.Reflection;

class Program3
{
    // Methods to get:

    public void MethodA(int i, int j) { }
```

```
    public void MethodA(int[] i) { }

    public unsafe void MethodA(int* i) { }

    public void MethodA(ref int r) {}

    // Method that takes an out parameter:
    public void MethodA(int i, out int o) { o = 100;}

  static void Main(string[] args)
  {
    MethodInfo mInfo;

    // Get MethodA(int i, int j)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        CallingConventions.Any,
        new Type[] { typeof(int), typeof(int) },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int[] i)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        CallingConventions.Any,
        new Type[] { typeof(int[]) },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int* i)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        CallingConventions.Any,
        new Type[] { typeof(int).MakePointerType() },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(ref int r)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        CallingConventions.Any,
        new Type[] { typeof(int).MakeByRefType() },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int i, out int o)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        CallingConventions.Any,
        new Type[] { typeof(int), typeof(int).MakeByRefType() },
```

```
            null);
        Console.WriteLine("Found method: {0}", mInfo);
    }
}
```

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

The following table shows what members of a base class are returned by the `GetXXX` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the

derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

The following BindingFlags filter flags can be used to define which methods to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public methods in the search.
- Specify `BindingFlags.NonPublic` to include nonpublic methods (that is, private, internal, and protected methods) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the methods declared on the Type, not methods that were simply inherited.

See System.Reflection.BindingFlags for more information.

> ⓘ **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> ⓘ **Note**

For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MethodInfo
- String
- BindingFlags
- Binder
- DefaultBinder
- CallingConventions
- ParameterModifier
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMethod(String, Int32, BindingFlags, Binder, Type[], ParameterModifier[])

Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, int
genericParameterCount, System.Reflection.BindingFlags bindingAttr,
System.Reflection.Binder? binder, Type[] types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the public method to get.

`genericParameterCount` Int32

The number of generic type parameters of the method.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through

COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified generic parameter count, argument types, modifiers and binding constraints, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in the `types` array is `null`.

ArgumentException

`genericParameterCount` is negative.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[])

Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, Type[] types, System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the method to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified requirements, if found; otherwise, `null`.

Implements

GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[])

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

Examples

The following example finds specific overloads of `MethodA`, specifying binding constraints and a variety of argument types.

> **Note**
>
> The Visual C# 2005 example requires the `/unsafe` compiler option.

C#

```
using System;
using System.Reflection;

class Program5
{
    // Methods to get:

    public void MethodA(int i, int j) { }

    public void MethodA(int[] i) { }

    public unsafe void MethodA(int* i) { }

    public void MethodA(ref int r) {}

    // Method that takes an out parameter.
    public void MethodA(int i, out int o) { o = 100; }

  static void Main(string[] args)
  {
    MethodInfo mInfo;

    // Get MethodA(int i, int j)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        new Type[] { typeof(int), typeof(int) },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int[] i)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        new Type[] { typeof(int[]) },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int* i)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        new Type[] { typeof(int).MakePointerType() },
        null);
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(ref int r)
    mInfo = typeof(Program).GetMethod("MethodA",
        BindingFlags.Public | BindingFlags.Instance,
        null,
        new Type[] { typeof(int).MakeByRefType() },
```

```
            null);
        Console.WriteLine("Found method: {0}", mInfo);

        // Get MethodA(int i, out int o)
        mInfo = typeof(Program).GetMethod("MethodA",
            BindingFlags.Public | BindingFlags.Instance,
            null,
            new Type[] { typeof(int), typeof(int).MakeByRefType() },
            null);
        Console.WriteLine("Found method: {0}", mInfo);
    }
}
```

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

The following BindingFlags filter flags can be used to define which methods to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public methods in the search.
- Specify `BindingFlags.NonPublic` to include nonpublic methods (that is, private, internal, and protected methods) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the methods declared on the Type, not methods that were simply inherited.

See System.Reflection.BindingFlags for more information.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MethodInfo
- String
- BindingFlags
- Binder
- DefaultBinder
- ParameterModifier
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8,

Product	Versions
Framework	4.8.1
.NET Standard	2.0, 2.1

GetMethod(String, Int32, Type[], ParameterModifier[])

Searches for the specified public method whose parameters match the specified generic parameter count, argument types and modifiers.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, int
genericParameterCount, Type[] types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the public method to get.

`genericParameterCount` Int32

The number of generic type parameters of the method.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the public method that matches the specified generic parameter count, argument types and modifiers, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in the `types` array is `null`.

ArgumentException

`genericParameterCount` is negative.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetMethod(String, BindingFlags)

Searches for the specified method, using the specified binding constraints.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the method to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

MethodInfo

An object representing the method that matches the specified requirements, if found; otherwise, `null`.

Implements

GetMethod(String, BindingFlags)

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

Examples

The following example gets the method that matches the specified binding flags.

C#

```
using System;
using System.Reflection;

class Program2
{
```

```
    // Method to get:
    public void MethodA() { }

    static void Main(string[] args)
    {

        // Get MethodA()
        MethodInfo mInfo = typeof(Program).GetMethod("MethodA",
            BindingFlags.Public | BindingFlags.Instance);
        Console.WriteLine("Found method: {0}", mInfo);
    }
}
```

Remarks

The following BindingFlags filter flags can be used to define which methods to include in the search:

- You must specify either BindingFlags.Instance or BindingFlags.Static in order to get a return.
- Specify BindingFlags.Public to include public methods in the search.
- Specify BindingFlags.NonPublic to include non-public methods (that is, private, internal, and protected methods) in the search.
- Specify BindingFlags.FlattenHierarchy to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- BindingFlags.IgnoreCase to ignore the case of `name`.
- BindingFlags.DeclaredOnly to search only the methods declared on the Type, not methods that were simply inherited.

See System.Reflection.BindingFlags for more information.

If a method is overloaded and more than one overload meets the constraints specified by the `bindingAttr` argument, the method throws an AmbiguousMatchException exception. In the following example, an exception is thrown because:

- The `TestClass` type has two public instance overloads of the `DisplayValue` method, `DisplayValue(String)` and `DisplayValue(String, Object[])`.
- The `TestClass` type has two public instance overloads of the `Equals` method, one of which is inherited from Object: `Equals(TestClass)` and `Equals(Object)`.

C#

```
using System;
using System.Reflection;

public class TestClass
{
    public void DisplayValue(String s)
    {
        Console.WriteLine(s);
    }

    public void DisplayValue(String s, params Object[] values)
    {
        Console.WriteLine(s, values);
    }

    public static bool Equals(TestClass t1, TestClass t2)
    {
        return Object.ReferenceEquals(t1, t2);
    }

    public bool Equals(TestClass t)
    {
        return Object.ReferenceEquals(this, t);
    }
}

public class Example1
{
    public static void Main()
    {
        Type t = typeof(TestClass);

        RetrieveMethod(t, "DisplayValue", BindingFlags.Public |
BindingFlags.Instance);

        RetrieveMethod(t, "Equals", BindingFlags.Public |
BindingFlags.Instance);

        RetrieveMethod(t, "Equals", BindingFlags.Public |
BindingFlags.Instance | BindingFlags.DeclaredOnly);

        RetrieveMethod(t, "Equals", BindingFlags.Public |
BindingFlags.Static);
    }
```

```
    private static void RetrieveMethod(Type t, String name, BindingFlags
flags)
    {
        try
        {
            MethodInfo m = t.GetMethod(name, flags);
            if (m != null)
            {
                Console.Write("{0}.{1}(", t.Name, m.Name);
                ParameterInfo[] parms = m.GetParameters();
                for (int ctr = 0; ctr < parms.Length; ctr++)
                {
                    Console.Write(parms[ctr].ParameterType.Name);
                    if (ctr < parms.Length - 1)
                        Console.Write(", ");
                }
                Console.WriteLine(")");
            }
            else
            {
                Console.WriteLine("Method not found");
            }
        }
        catch (AmbiguousMatchException)
        {
            Console.WriteLine("The following duplicate matches were
found:");
            MethodInfo[] methods = t.GetMethods(flags);
            foreach (var method in methods)
            {
                if (method.Name != name) continue;

                Console.Write("   {0}.{1}(", t.Name, method.Name);
                ParameterInfo[] parms = method.GetParameters();
                for (int ctr = 0; ctr < parms.Length; ctr++)
                {
                    Console.Write(parms[ctr].ParameterType.Name);
                    if (ctr < parms.Length - 1)
                        Console.Write(", ");
                }
                Console.WriteLine(")");
            }
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//       The following duplicate matches were found:
//          TestClass.DisplayValue(String)
//          TestClass.DisplayValue(String, Object[])
//
//       The following duplicate matches were found:
//          TestClass.Equals(TestClass)
//          TestClass.Equals(Object)
//
```

```
//       TestClass.Equals(TestClass)
//
//       TestClass.Equals(TestClass, TestClass)
```

You can do one of the following to retrieve a specific method:

- Change the binding constraints. In the previous example, attempting to retrieve a public instance `Equals` method that is declared by the type and not inherited successfully retrieves `Equals(TestClass)`.
- Call an overload of the GetMethod method that includes a `types` parameter which defines the types of the method's parameters.
- Call the GetMethods(BindingFlags) method to retrieve an array containing all of the methods belonging to a type that have the specified binding attributes. You can then iterate it to identify the duplicate methods named `name`. This approach is illustrated in the previous example's handler for the AmbiguousMatchException exception.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MethodInfo
- String
- BindingFlags
- DefaultBinder
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMethod(String, BindingFlags, Type[])

Searches for the specified method whose parameters match the specified argument types, using the specified binding constraints.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name,
System.Reflection.BindingFlags bindingAttr, Type[] types);
```

Parameters

`name` String

The string containing the name of the method to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted. -or- Default to return `null`.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get. -or- An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

Returns

MethodInfo

An object representing the method that matches the specified requirements, if found; otherwise, `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

GetMethod(String, Int32, Type[])

Searches for the specified public method whose parameters match the specified generic parameter count and argument types.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, int
genericParameterCount, Type[] types);
```

Parameters

`name` String

The string containing the name of the public method to get.

`genericParameterCount` Int32

The number of generic type parameters of the method.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

Returns

MethodInfo

An object representing the public method whose parameters match the specified generic parameter count and argument types, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in the `types` array is `null`.

ArgumentException

`genericParameterCount` is negative.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetMethod(String, Type[])

Searches for the specified public method whose parameters match the specified argument types.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, Type[]
types);
```

Parameters

name String

The string containing the name of the public method to get.

types Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

Returns

MethodInfo

An object representing the public method whose parameters match the specified argument types, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and specified parameters.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

Examples

The following example finds specific overloads of `MethodA`, specifying a variety of argument types.

ⓘ **Note**

The Visual C# 2005 example requires the `/unsafe` compiler option.

C#

```
using System;
using System.Reflection;

class Program4
{
    // Methods to get:

    public void MethodA(int i, int j) { }

    public void MethodA(int[] i) { }

    public unsafe void MethodA(int* i) { }

    public void MethodA(ref int r) {}

    // Method that takes an out parameter:
    public void MethodA(int i, out int o) { o = 100;}

  static void Main(string[] args)
  {
    MethodInfo mInfo;

    // Get MethodA(int i, int i)
    mInfo = typeof(Program).GetMethod("MethodA",
        new Type[] { typeof(int), typeof(int) });
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int[] i)
    mInfo = typeof(Program).GetMethod("MethodA",
        new Type[] { typeof(int[]) });
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int* i)
    mInfo = typeof(Program).GetMethod("MethodA",
        new Type[] { typeof(int).MakePointerType() });
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(ref int r)
    mInfo = typeof(Program).GetMethod("MethodA",
        new Type[] { typeof(int).MakeByRefType() });
    Console.WriteLine("Found method: {0}", mInfo);

    // Get MethodA(int i, out int o)
    mInfo = typeof(Program).GetMethod("MethodA",
        new Type[] { typeof(int), typeof(int).MakeByRefType() });
    Console.WriteLine("Found method: {0}", mInfo);
```

```
    }
  }
```

The following example retrieves MethodInfo objects that represent the `Add` methods of a non-generic type (the ArrayList class), an open generic type (the List<T> class), and a closed generic type (the `List(Of String)` type.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;

public class Example
{
   public static void Main()
   {
      // Get a Type object that represents a non-generic type.
      GetAddMethod(typeof(ArrayList));

      var list = new List<String>();
      // Get a Type object that represents a constructed generic type.
      Type closed = list.GetType();
      GetAddMethod(closed);

      // Get a Type object that represents an open generic type.
      Type open = typeof(List<>);
      GetAddMethod(open);
   }

   private static void GetAddMethod(Type typ)
   {
      MethodInfo method;
      // Determine if this is a generic type.
      if (typ.IsGenericType) {
         // Is it an open generic type?
         if (typ.ContainsGenericParameters)
            method = typ.GetMethod("Add", typ.GetGenericArguments());
         // Get closed generic type arguments.
         else
            method = typ.GetMethod("Add", typ.GenericTypeArguments);
      }
      // This is not a generic type.
      else {
         method = typ.GetMethod("Add", new Type[] { typeof(Object) } );
      }

      // Test if an Add method was found.
      if (method == null) {
         Console.WriteLine("No Add method found.");
         return;
```

```
        }

        Type t = method.ReflectedType;
        Console.Write("{0}.{1}.{2}(", t.Namespace, t.Name, method.Name);
        ParameterInfo[] parms = method.GetParameters();
        for (int ctr = 0; ctr < parms.Length; ctr++)
           Console.Write("{0}{1}", parms[ctr].ParameterType.Name,
                         ctr < parms.Length - 1 ? ", " : "");

        Console.WriteLine(")");
    }
}
// The example displays the following output:
//       System.Collections.ArrayList.Add(Object)
//       System.Collections.Generic.List`1.Add(String)
//       System.Collections.Generic.List`1.Add(T)
```

The example defines a `GetAddMethod` method that retrieves the appropriate MethodInfo object. To provide the `types` argument for an open generic type, it calls the Type.GetGenericArguments method. To provide the `types` argument for a closed generic type, it retrieves the value of the Type.GenericTypeArguments property.

Remarks

The search for `name` is case-sensitive. The search includes public static and public instance methods.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> **Note**
>
> The `name` parameter cannot include type arguments. For example, the C# code `GetMethod("MyGenericMethod<int>")` searches for a method with the text name "`MyGenericMethod<int>`", rather than for a method named `MyGenericMethod` that

has one generic argument of type `int`. Instead, use `GetMethod("MyGenericMethod")` with the appropriate parameter in the `types` array.

See also

- MethodInfo
- DefaultBinder
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMethod(String)

Searches for the public method with the specified name.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name);
```

Parameters

name String

The string containing the name of the public method to get.

Returns

MethodInfo

An object that represents the public method with the specified name, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one method is found with the specified name.

ArgumentNullException

`name` is `null`.

Examples

The following example gets a method named `MethodA`.

C#

```
using System;
using System.Reflection;

class Program
{

    // Method to get:
    public void MethodA() { }

    static void Main(string[] args)
    {

        // Get MethodA()
        MethodInfo mInfo = typeof(Program).GetMethod("MethodA");
        Console.WriteLine("Found method: {0}", mInfo);
    }
}
```

Remarks

The search for `name` is case-sensitive. The search includes public static and public instance methods.

If a method is overloaded and has more than one public method, the GetMethod(String) method throws an AmbiguousMatchException exception. In the following example, an exception is thrown because there is more than one public overload of the Int32.ToString method. On the other hand, because the

`Person.ToString` method overrides Object.ToString and therefore is not overloaded, the GetMethod(String) method is able to retrieve the MethodInfo object.

C#

```
using System;
using System.Reflection;

public class Person
{
    public String FirstName;
    public String LastName;

    public override String ToString()
    {
        return (FirstName + " " + LastName).Trim();
    }
}

public class Example2
{
    public static void Main()
    {
        Type t = typeof(Person);
        RetrieveMethod(t, "ToString");

        t = typeof(Int32);
        RetrieveMethod(t, "ToString");
    }

    private static void RetrieveMethod(Type t, String name)
    {
        try
        {
            MethodInfo m = t.GetMethod(name);
            if (m != null)
                Console.WriteLine("{0}.{1}: {2} method",
m.ReflectedType.Name,
                                  m.Name, m.IsStatic ? "Static" :
"Instance");
            else
                Console.WriteLine("{0}.ToString method not found",
t.Name);
        }
        catch (AmbiguousMatchException)
        {
            Console.WriteLine("{0}.{1} has multiple public overloads.",
                              t.Name, name);
        }
    }
}
// The example displays the following output:
```

```
//       Person.ToString: Instance method
//       Int32.ToString has multiple public overloads.
```

You can do one of the following to retrieve a specific method:

- Call the GetMethod(String, BindingFlags) method and specify a `bindingAttr` argument that uniquely identifies the method. For example, if the exception is thrown because a type has a static and an instance overload, you can specify a `bindingAttr` argument of BindingFlags.Instance `Or` BindingFlags.Instance.
- Call an overload of the GetMethod method that includes a `types` parameter which defines the types of the method's parameters.
- Call the GetMethods() method to retrieve an array containing all of the public methods belonging to a type. You can then iterate it to identify the duplicate methods named `name`.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMember("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`.

See also

- MethodInfo
- String
- DefaultBinder
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMethod(String, Type[], ParameterModifier[])

Searches for the specified public method whose parameters match the specified argument types and modifiers.

C#

```
public System.Reflection.MethodInfo? GetMethod (string name, Type[]
types, System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the public method to get.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of Type objects (as provided by the EmptyTypes field) to get a method that takes no parameters.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. To be only used when calling through COM interop, and only parameters that are passed by reference are handled. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the public method that matches the specified requirements, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and specified parameters.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

The search for `name` is case-sensitive. The search includes public static and public instance methods.

> **Note**

You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the MethodInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

> **Note**
>
> For generic methods, do not include the type arguments in `name`. For example, the C# code `GetMethod("MyMethod<int>")` searches for a member with the text name "`MyMethod<int>`", rather than for a method named `MyMethod` that has one generic argument of type `int`. Instead, use `GetMethod("MyMethod")` with the appropriate parameter in the `types` array.

See also

- MethodInfo
- String
- DefaultBinder
- ParameterModifier
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetMethodImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

GetMethodImpl(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

GetMethodImpl(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

When overridden in a derived class, searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.

C#

```
protected virtual System.Reflection.MethodInfo? GetMethodImpl (string
name, int genericParameterCount, System.Reflection.BindingFlags
bindingAttr, System.Reflection.Binder? binder,
System.Reflection.CallingConventions callConvention, Type[]? types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the method to get.

`genericParameterCount` Int32

The number of generic type parameters of the method.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and what process cleans up the stack.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a method that takes no parameters.

-or-

`null`. If `types` is `null`, arguments are not matched.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified generic parameter count, argument types, modifiers, binding constraints and calling convention, if found; otherwise, `null`.

Exceptions

NotSupportedException

The method needs to be overriden and called in a derived class.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

When overridden in a derived class, searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

C#

```
protected abstract System.Reflection.MethodInfo? GetMethodImpl (string
name, System.Reflection.BindingFlags bindingAttr,
System.Reflection.Binder? binder, System.Reflection.CallingConventions
```

```
callConvention, Type[]? types, System.Reflection.ParameterModifier[]?
modifiers);
```

Parameters

`name` String

The string containing the name of the method to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`callConvention` CallingConventions

The object that specifies the set of rules to use regarding the order and layout of arguments, how the return value is passed, what registers are used for arguments, and what process cleans up the stack.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the method to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a method that takes no parameters.

-or-

`null`. If `types` is `null`, arguments are not matched.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

MethodInfo

An object representing the method that matches the specified requirements, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one method is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

NotSupportedException

The current type is a TypeBuilder or GenericTypeParameterBuilder.

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

If `types` is `null`, arguments are not matched.

The following BindingFlags filter flags can be used to define which methods to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public methods in the search.
- Specify `BindingFlags.NonPublic` to include non-public methods (that is, private, internal, and protected methods) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the methods declared on the Type, not methods that were simply inherited.

See System.Reflection.BindingFlags for more information.

See also

- MethodInfo
- String
- BindingFlags
- Binder
- DefaultBinder
- CallingConventions
- ParameterModifier
- GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetMethods Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the methods of the current Type.

Overloads

Expand table

GetMethods(BindingFlags)	When overridden in a derived class, searches for the methods defined for the current Type, using the specified binding constraints.
GetMethods()	Returns all the public methods of the current Type.

GetMethods(BindingFlags)

When overridden in a derived class, searches for the methods defined for the current Type, using the specified binding constraints.

C#

```
public abstract System.Reflection.MethodInfo[] GetMethods
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

bindingAttr BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

MethodInfo[]

An array of MethodInfo objects representing all methods defined for the current Type that match the specified binding constraints.

-or-

An empty array of type MethodInfo, if no methods are defined for the current Type, or if none of the defined methods match the binding constraints.

Implements

GetMethods(BindingFlags)

Examples

The following example creates a class with two public methods and one protected method, creates a `Type` object corresponding to `MyTypeClass`, gets all public and non-public methods, and displays their names.

C#

```
using System;
using System.Reflection;
using System.Reflection.Emit;

        // Create a class having two public methods and one protected
method.
public class MyTypeClass
{
    public void MyMethods()
    {
    }
    public int MyMethods1()
    {
        return 3;
    }
    protected String MyMethods2()
    {
        return "hello";
    }
}
public class TypeMain
{
    public static void Main()
    {
```

```
        Type myType =(typeof(MyTypeClass));
        // Get the public methods.
        MethodInfo[] myArrayMethodInfo =
myType.GetMethods(BindingFlags.Public|BindingFlags.Instance|BindingFlags.
DeclaredOnly);
        Console.WriteLine("\nThe number of public methods is {0}.",
myArrayMethodInfo.Length);
        // Display all the methods.
        DisplayMethodInfo(myArrayMethodInfo);
        // Get the nonpublic methods.
        MethodInfo[] myArrayMethodInfo1 =
myType.GetMethods(BindingFlags.NonPublic|BindingFlags.Instance|BindingFla
gs.DeclaredOnly);
        Console.WriteLine("\nThe number of protected methods is {0}.",
myArrayMethodInfo1.Length);
        // Display information for all methods.
        DisplayMethodInfo(myArrayMethodInfo1);
    }
    public static void DisplayMethodInfo(MethodInfo[] myArrayMethodInfo)
    {
        // Display information for all methods.
        for(int i=0;i<myArrayMethodInfo.Length;i++)
        {
            MethodInfo myMethodInfo = (MethodInfo)myArrayMethodInfo[i];
            Console.WriteLine("\nThe name of the method is {0}.",
myMethodInfo.Name);
        }
    }
}
```

Remarks

For the `GetMethods(BindingFlags)` overload to successfully retrieve method information, the `bindingAttr` argument must include at least one of BindingFlags.Instance and BindingFlags.Static, along with at least one of BindingFlags.NonPublic and BindingFlags.Public.

The following BindingFlags filter flags can be used to define which methods to include in the search:

- Specify `BindingFlags.Instance` to include instance methods.
- Specify `BindingFlags.Static` to include static methods.
- Specify `BindingFlags.Public` to include public methods in the search.
- Specify `BindingFlags.NonPublic` to include non-public methods (that is, private, internal, and protected methods) in the search. Only protected and internal

methods on base classes are returned; private methods on base classes are not returned.

- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.
- Specify `BindingFlags.Default` alone to return an empty MethodInfo array.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the methods declared on the Type, not methods that were simply inherited.

See System.Reflection.BindingFlags for more information.

In .NET 6 and earlier versions, the GetMethods method does not return methods in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which methods are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

If the current Type represents a constructed generic type, this method returns the MethodInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the methods of Object if there is no class constraint.

See also

- MethodInfo
- BindingFlags
- DefaultBinder
- GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetMethods()

Returns all the public methods of the current Type.

```
C#
public System.Reflection.MethodInfo[] GetMethods ();
```

Returns

MethodInfo[]

An array of MethodInfo objects representing all the public methods defined for the current Type.

-or-

An empty array of type MethodInfo, if no public methods are defined for the current Type.

Remarks

In .NET 6 and earlier versions, the GetMethods method does not return methods in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which methods are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

Constructors are not included in the array of methods returned by this call. Make a separate call to `GetConstructors()` to get the constructor methods.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

> **Note**
>
> You cannot omit parameters when looking up constructors and methods. You can only omit parameters when invoking.

If the current Type represents a constructed generic type, this method returns the MethodInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the methods of the class constraint, or the

methods of Object if there is no class constraint.

See also

- MethodInfo
- GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetNestedType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific type nested within the current Type.

Overloads

Expand table

GetNestedType(String)	Searches for the public nested type with the specified name.
GetNestedType(String, BindingFlags)	When overridden in a derived class, searches for the specified nested type, using the specified binding constraints.

GetNestedType(String)

Searches for the public nested type with the specified name.

C#

```
public Type? GetNestedType (string name);
```

Parameters

name String

The string containing the name of the nested type to get.

Returns

Type

An object representing the public nested type with the specified name, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

Remarks

The search for `name` is case-sensitive.

Use the simple name of the nested class for `name`. Do not qualify it with the name of the outer class. For a generic nested class, use the mangled name - that is, append a grave accent and the number of generic arguments. For example, use the string "Inner`1" to get the generic nested class `Inner<T>` (`Inner(Of T)` in Visual Basic). Do not include language-specific syntax for type parameters.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the nested types of the class constraint.

If a nested type is generic, this method returns its generic type definition. This is true even if the enclosing generic type is a closed constructed type.

> ⓘ **Note**
>
> If the current **Type** represents a generic type defined in C#, Visual Basic, or C++, its nested types are all generic even if they have no generic parameters of their own. This is not necessarily true of nested types defined in dynamic assemblies or compiled with the **Ilasm.exe (IL Assembler)**.

For information on nested generic types, and on constructing nested generic types from their generic type definitions, see MakeGenericType.

See also

- String
- DefaultBinder
- GetNestedTypes()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetNestedType(String, BindingFlags)

When overridden in a derived class, searches for the specified nested type, using the specified binding constraints.

```
C#

public abstract Type? GetNestedType (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the nested type to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

Type

An object representing the nested type that matches the specified requirements, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`name` is `null`.

Remarks

Use the simple name of the nested class for `name`. Do not qualify it with the name of the outer class. For a generic nested class, use the mangled name - that is, append a grave accent and the number of generic parameters. For example, use the string "Inner`1" to get the generic nested class `Inner<T>` (`Inner(Of T)` in Visual Basic). Do not include language-specific syntax for type parameters.

The following BindingFlags filter flags can be used to define which nested types to include in the search:

- You must specify either BindingFlags.Public or BindingFlags.NonPublic to get a return.
- Specify BindingFlags.Public to include public nested types in the search.
- Specify BindingFlags.NonPublic to include non-public nested types (that is, private, internal, and protected nested types) in the search.

This method returns only the nested types of the current type. It does not search the base classes of the current type. To find types that are nested in base classes, you must walk the inheritance hierarchy, calling GetNestedType at each level.

BindingFlags.Instance and BindingFlags.Static are ignored.

Calling this method with only the BindingFlags.Public flag or only the BindingFlags.NonPublic flag will return the specified nested types and does not require any other flags.

See System.Reflection.BindingFlags for more information.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the nested types of the class constraint.

If a nested type is generic, this method returns its generic type definition. This is true even if the enclosing generic type is a closed constructed type.

> **Note**
>
> If the current **Type** represents a generic type defined in C#, Visual Basic, or C++, its nested types are all generic even if they have no generic parameters of their own. This is not necessarily true of nested types defined in dynamic assemblies or compiled with the **Ilasm.exe (IL Assembler)**.

For information on nested generic types, and on constructing nested generic types from their generic type definitions, see MakeGenericType.

See also

- String
- BindingFlags
- DefaultBinder

- GetNestedTypes()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetNestedTypes Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the types nested within the current Type.

Overloads

Expand table

GetNestedTypes()	Returns the public types nested in the current Type.
GetNestedTypes(BindingFlags)	When overridden in a derived class, searches for the types nested in the current Type, using the specified binding constraints.

GetNestedTypes()

Returns the public types nested in the current Type.

C#

```
public Type[] GetNestedTypes ();
```

Returns

Type[]

An array of Type objects representing the public types nested in the current Type (the search is not recursive), or an empty array of type Type if no public types are nested in the current Type.

Examples

The following example defines a nested class and a `struct` in `MyClass`, and then obtains objects of the nested types using the type of `MyClass`.

C#

```
using System;
using System.Reflection;
public class MyClass
{
    public class NestClass
    {
        public static int myPublicInt=0;
    }
    public struct NestStruct
    {
        public static int myPublicInt=0;
    }
}

public class MyMainClass
{
    public static void Main()
    {
        try
        {
            // Get the Type object corresponding to MyClass.
            Type myType=typeof(MyClass);
            // Get an array of nested type objects in MyClass.
            Type[] nestType=myType.GetNestedTypes();
            Console.WriteLine("The number of nested types is {0}.",
nestType.Length);
            foreach(Type t in nestType)
                Console.WriteLine("Nested type is {0}.", t.ToString());
        }
        catch(Exception e)
        {
            Console.WriteLine("Error"+e.Message);
        }
    }
}
```

Remarks

In .NET 6 and earlier versions, the GetNestedTypes method does not return types in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which types are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

Only the public types immediately nested in the current type are returned; the search is not recursive.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the nested types of the class constraint.

If a nested type is generic, this method returns its generic type definition. This is true even if the enclosing generic type is a closed constructed type.

> **Note**
>
> If the current **Type** represents a generic type defined in C#, Visual Basic, or C++, its nested types are all generic even if they have no generic parameters of their own. This is not necessarily true of nested types defined in dynamic assemblies or compiled with the **Ilasm.exe (IL Assembler)**.

For information on nested generic types, and on constructing nested generic types from their generic type definitions, see MakeGenericType.

See also

- GetNestedType(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetNestedTypes(BindingFlags)

When overridden in a derived class, searches for the types nested in the current Type, using the specified binding constraints.

C#

```
public abstract Type[] GetNestedTypes (System.Reflection.BindingFlags
bindingAttr);
```

Parameters

bindingAttr	BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

Type[]

An array of Type objects representing all the types nested in the current Type that match the specified binding constraints (the search is not recursive), or an empty array of type Type, if no nested types are found that match the binding constraints.

Examples

The following example creates two nested public classes and two nested protected classes, and displays information for classes that match the specified binding constraints.

C#

```
using System;
using System.Reflection;

// Create a class with 2 nested public and 2 nested protected classes.
public class MyTypeClass
{
    public class Myclass1
    {
    }

    public class Myclass2
    {
    }

    protected class MyClass3
    {
    }

    protected class MyClass4
    {
    }
}

public class TypeMain
{
    public static void Main()
```

```
    {
        Type myType = (typeof(MyTypeClass));
        // Get the public nested classes.
        Type[] myTypeArray = myType.GetNestedTypes(BindingFlags.Public);
        Console.WriteLine("The number of nested public classes is {0}.",
myTypeArray.Length);
        // Display all the public nested classes.
        DisplayTypeInfo(myTypeArray);
        Console.WriteLine();

        // Get the nonpublic nested classes.
        Type[] myTypeArray1 =
myType.GetNestedTypes(BindingFlags.NonPublic|BindingFlags.Instance);
        Console.WriteLine("The number of nested protected classes is
{0}.", myTypeArray1.Length);
        // Display all the nonpublic nested classes.
        DisplayTypeInfo(myTypeArray1);
    }

    public static void DisplayTypeInfo(Type[] myArrayType)
    {
        // Display the information for all the nested classes.
        foreach (var t in myArrayType)
            Console.WriteLine("The name of the nested class is {0}.",
t.FullName);
    }
}
// The example displays the following output:
//       The number of public nested classes is 2.
//       The name of the nested class is MyTypeClass+Myclass1.
//       The name of the nested class is MyTypeClass+Myclass2.
//
//       The number of protected nested classes is 2.
//       The name of the nested class is MyTypeClass+MyClass3.
//       The name of the nested class is MyTypeClass+MyClass4.
```

Remarks

The search for nested types is not recursive.

In .NET 6 and earlier versions, the GetNestedTypes method does not return types in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which types are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following BindingFlags filter flags can be used to define which nested types to include in the search:

- You must specify either BindingFlags.Public or BindingFlags.NonPublic to get a return.
- Specify BindingFlags.Public to include public nested types in the search.
- Specify BindingFlags.NonPublic to include non-public nested types (that is, private, internal, and protected nested types) in the search.

This method returns only the nested types of the current type. It does not search the base classes of the current type. To find types that are nested in base classes, you must walk the inheritance hierarchy, calling GetNestedTypes at each level.

BindingFlags.Instance and BindingFlags.Static are ignored.

Calling this method with only the BindingFlags.Public flag or only the BindingFlags.NonPublic flag will return the specified nested types and does not require any other flags.

See System.Reflection.BindingFlags for more information.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the nested types of the class constraint.

If a nested type is generic, this method returns its generic type definition. This is true even if the enclosing generic type is a closed constructed type.

> **Note**
>
> If the current **Type** represents a generic type defined in C#, Visual Basic, or C++, its nested types are all generic even if they have no generic parameters of their own. This is not necessarily true of nested types defined in dynamic assemblies or compiled with the **Ilasm.exe (IL Assembler)**.

For information on nested generic types, and on constructing nested generic types from their generic type definitions, see MakeGenericType.

See also

- BindingFlags
- DefaultBinder
- GetNestedType(String)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetOptionalCustomModifiers Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the optional custom modifiers of the current Type.

```
C#

public virtual Type[] GetOptionalCustomModifiers ();
```

Returns

Type[]

An array of Type objects that identify the optional custom modifiers of the current Type.

-or-

An empty array of type Type, if the current Type has no custom modifiers.

-or-

An empty array of type Type, if the current Type is not a modified Type. A modified Type is obtained from GetModifiedFieldType(), GetModifiedPropertyType(), or GetModifiedParameterType().

Remarks

The GetOptionalCustomModifiers and GetRequiredCustomModifiers methods are provided for designers of managed compilers. For more information on custom modifiers, see classes in the System.Runtime.CompilerServices namespace. Also see the metadata specification in Partition II of the Common Language Infrastructure (CLI) documentation.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetProperties Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the properties of the current Type.

Overloads

Expand table

GetProperties()	Returns all the public properties of the current Type.
GetProperties(BindingFlags)	When overridden in a derived class, searches for the properties of the current Type, using the specified binding constraints.

GetProperties()

Returns all the public properties of the current Type.

C#

```
public System.Reflection.PropertyInfo[] GetProperties ();
```

Returns

PropertyInfo[]

An array of PropertyInfo objects representing all public properties of the current Type.

-or-

An empty array of type PropertyInfo, if the current Type does not have public properties.

Examples

The following example demonstrates the use of the `GetProperties` method.

C#

```
PropertyInfo[] myPropertyInfo;
// Get the properties of 'Type' class object.
myPropertyInfo = Type.GetType("System.Type").GetProperties();
Console.WriteLine("Properties of System.Type are:");
for (int i = 0; i < myPropertyInfo.Length; i++)
{
    Console.WriteLine(myPropertyInfo[i].ToString());
}
```

Remarks

Calling this overload is equivalent to calling the GetProperties(BindingFlags) overload with a `bindingAttr` argument equal to `BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public` in C# and `BindingFlags.Instance Or BindingFlags.Static Or BindingFlags.Public` in Visual Basic. It returns all public instance and static properties, both those defined by the type represented by the current Type object as well as those inherited from its base types.

A property is considered public to reflection if it has at least one accessor that is public. Otherwise the property is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

In .NET 6 and earlier versions, the GetProperties method does not return properties in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which properties are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No

Member Type	Static	Non-Static
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

If the current Type represents a constructed generic type, this method returns the PropertyInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo
- GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperties(BindingFlags)

When overridden in a derived class, searches for the properties of the current Type, using the specified binding constraints.

C#

```
public abstract System.Reflection.PropertyInfo[] GetProperties
(System.Reflection.BindingFlags bindingAttr);
```

Parameters

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return an empty array.

Returns

PropertyInfo[]

An array of objects representing all properties of the current Type that match the specified binding constraints.

-or-

An empty array of type PropertyInfo, if the current Type does not have properties, or if none of the properties match the binding constraints.

Implements

GetProperties(BindingFlags)

Examples

The following example defines a class named `PropertyClass` that includes six properties: two are public, one is private, one is protected, one is internal (`Friend` in Visual Basic), and one is protected internal (`Protected Friend` in Visual Basic). It then displays some basic property information (the property name and type, whether it is read/write, and the visibility of its `get` and `set` accessors) for the properties that match the specified binding constraints.

C#

```
using System;
using System.Reflection;

// Create a class having six properties.
public class PropertyClass
{
    public String Property1
    {
        get { return "hello"; }
    }

    public String Property2
    {
        get { return "hello"; }
    }

    protected String Property3
    {
        get { return "hello"; }
    }

    private Int32 Property4
    {
        get { return 32; }
    }

    internal String Property5
    {
       get { return "value"; }
    }

    protected internal String Property6
    {
       get { return "value"; }
    }
}

public class Example
{
    public static void Main()
```

```
    {
        Type t = typeof(PropertyClass);
        // Get the public properties.
        PropertyInfo[] propInfos =
t.GetProperties(BindingFlags.Public|BindingFlags.Instance);
        Console.WriteLine("The number of public properties: {0}.\n",
                          propInfos.Length);
        // Display the public properties.
        DisplayPropertyInfo(propInfos);

        // Get the nonpublic properties.
        PropertyInfo[] propInfos1 =
t.GetProperties(BindingFlags.NonPublic|BindingFlags.Instance);
        Console.WriteLine("The number of non-public properties: {0}.\n",
                          propInfos1.Length);
        // Display all the nonpublic properties.
        DisplayPropertyInfo(propInfos1);
    }

    public static void DisplayPropertyInfo(PropertyInfo[] propInfos)
    {
        // Display information for all properties.
        foreach (var propInfo in propInfos) {
            bool readable = propInfo.CanRead;
            bool writable = propInfo.CanWrite;

            Console.WriteLine("   Property name: {0}", propInfo.Name);
            Console.WriteLine("   Property type: {0}",
propInfo.PropertyType);
            Console.WriteLine("   Read-Write:    {0}", readable &
writable);
            if (readable) {
               MethodInfo getAccessor = propInfo.GetMethod;
               Console.WriteLine("   Visibility:    {0}",
                                 GetVisibility(getAccessor));
            }
            if (writable) {
               MethodInfo setAccessor = propInfo.SetMethod;
               Console.WriteLine("   Visibility:    {0}",
                                 GetVisibility(setAccessor));
            }
            Console.WriteLine();
        }
    }

    public static String GetVisibility(MethodInfo accessor)
    {
       if (accessor.IsPublic)
          return "Public";
       else if (accessor.IsPrivate)
          return "Private";
       else if (accessor.IsFamily)
          return "Protected";
       else if (accessor.IsAssembly)
          return "Internal/Friend";
```

```
        else
           return "Protected Internal/Friend";
    }
}
// The example displays the following output:
//       The number of public properties: 2.
//
//          Property name: Property1
//          Property type: System.String
//          Read-Write:    False
//          Visibility:    Public
//
//          Property name: Property2
//          Property type: System.String
//          Read-Write:    False
//          Visibility:    Public
//
//       The number of non-public properties: 4.
//
//          Property name: Property3
//          Property type: System.String
//          Read-Write:    False
//          Visibility:    Protected
//
//          Property name: Property4
//          Property type: System.Int32
//          Read-Write:    False
//          Visibility:    Private
//
//          Property name: Property5
//          Property type: System.String
//          Read-Write:    False
//          Visibility:    Internal/Friend
//
//          Property name: Property6
//          Property type: System.String
//          Read-Write:    False
//          Visibility:    Protected Internal/Friend
```

Remarks

For the `GetProperties(BindingFlags)` overload to successfully retrieve property information, the `bindingAttr` argument must include at least one of BindingFlags.Instance and BindingFlags.Static, along with at least one of BindingFlags.NonPublic and BindingFlags.Public.

The following BindingFlags filter flags can be used to define which properties to include in the search:

- Specify `BindingFlags.Instance` to include instance methods.

- Specify `BindingFlags.Static` to include static methods.
- Specify `BindingFlags.Public` to include public properties in the search. A property is considered public to reflection if it has at least one accessor that is public.
- Specify `BindingFlags.NonPublic` to include non-public properties (that is, private, internal, and protected properties) in the search. Only protected and internal properties on base classes are returned; private properties on base classes are not returned.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.
- Specify `BindingFlags.Default` alone to return an empty PropertyInfo array.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.DeclaredOnly` to search only the properties declared on the Type, not properties that were simply inherited.

See System.Reflection.BindingFlags for more information.

In .NET 6 and earlier versions, the GetProperties method does not return properties in a particular order, such as alphabetical or declaration order. Your code must not depend on the order in which properties are returned, because that order varies. However, starting with .NET 7, the ordering is deterministic based upon the metadata ordering in the assembly.

If the current Type represents a constructed generic type, this method returns the PropertyInfo objects with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo
- BindingFlags
- DefaultBinder
- GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetProperty Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a specific property of the current Type.

Overloads

Expand table

GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])	Searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetProperty(String, Type, Type[], ParameterModifier[])	Searches for the specified public property whose parameters match the specified argument types and modifiers.
GetProperty(String, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetProperty(String, Type, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetProperty(String, BindingFlags)	Searches for the specified property, using the specified binding constraints.
GetProperty(String)	Searches for the public property with the specified name.
GetProperty(String, Type)	Searches for the public property with the specified name and return type.

GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])

Searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.

```
C#

public System.Reflection.PropertyInfo? GetProperty (string name,
System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, Type? returnType, Type[] types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the property to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`returnType` Type

The return type of the property.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the indexed property to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a property that is not indexed.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

PropertyInfo

An object representing the property that matches the specified requirements, if found; otherwise, `null`.

Implements

GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

NullReferenceException

An element of `types` is `null`.

Remarks

For more information about this API, see Supplemental API remarks for Type.GetProperty.

See also

- PropertyInfo
- String
- BindingFlags
- Binder
- DefaultBinder
- ParameterModifier
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String, Type, Type[], ParameterModifier[])

Searches for the specified public property whose parameters match the specified argument types and modifiers.

C#

```
public System.Reflection.PropertyInfo? GetProperty (string name, Type?
returnType, Type[] types, System.Reflection.ParameterModifier[]?
modifiers);
```

Parameters

name String

The string containing the name of the public property to get.

`returnType` Type

The return type of the property.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the indexed property to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a property that is not indexed.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

PropertyInfo

An object representing the public property that matches the specified requirements, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified argument types and modifiers.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

NullReferenceException

An element of `types` is `null`.

Examples

The following example obtains a `Type` object corresponding to `MyPropertyClass`, and the indexed property of this class is retrieved using the arguments passed to the `GetProperty` method.

C#

```
using System;
using System.Reflection;

public class MyPropertyClass
{
    private int [,] myPropertyArray = new int[10,10];
    // Declare an indexer.
    public int this [int i,int j]
    {
        get
        {
            return myPropertyArray[i,j];
        }
        set
        {
            myPropertyArray[i,j] = value;
        }
    }
}

public class MyTypeClass
{
    public static void Main()
    {
        try
        {
            Type myType=typeof(MyPropertyClass);
            Type[] myTypeArray = new Type[2];
            // Create an instance of the Type array representing the
number, order
            // and type of the parameters for the property.
            myTypeArray.SetValue(typeof(int),0);
            myTypeArray.SetValue(typeof(int),1);
```

```
            // Search for the indexed property whose parameters match the
            // specified argument types and modifiers.
            PropertyInfo myPropertyInfo = myType.GetProperty("Item",
                typeof(int),myTypeArray,null);
            Console.WriteLine(myType.FullName + "." + myPropertyInfo.Name
+
                " has a property type of " +
myPropertyInfo.PropertyType);
         }
        catch(Exception ex)
        {
            Console.WriteLine("An exception occurred " + ex.Message);
        }
    }
}
```

Remarks

A property is considered public to reflection if it has at least one accessor that is public. Otherwise the property is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

The search for `name` is case-sensitive. The search includes public static and public instance properties.

If the current Type represents a constructed generic type, this method returns the PropertyInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo
- String
- DefaultBinder
- ParameterModifier
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])

- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String, Type[])

Searches for the specified public property whose parameters match the specified argument types.

C#

```
public System.Reflection.PropertyInfo? GetProperty (string name, Type[] types);
```

Parameters

`name` String

The string containing the name of the public property to get.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the indexed property to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a property that is not indexed.

Returns

PropertyInfo

An object representing the public property whose parameters match the specified argument types, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified argument types.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

ArgumentException

`types` is multidimensional.

NullReferenceException

An element of `types` is `null`.

Examples

The following example retrieves the `Type` object of a user-defined class, retrieves the property of that class, and displays the property name and type of the property as specified by the arguments passed to `GetProperty`.

C#

```
using System;
using System.Reflection;

class MyClass3
{
    private int [,] myArray = {{1,2},{3,4}};
    // Declare an indexer.
    public int this [int i,int j]
    {
        get
        {
            return myArray[i,j];
        }
        set
```

```
        {
            myArray[i,j] = value;
        }
    }
}

public class MyTypeClass3
{
    public static void Main(string[] args)
    {
        try
        {
            // Get the Type object.
            Type myType=typeof(MyClass3);
            Type[] myTypeArr = new Type[2];
            // Create an instance of a Type array.
            myTypeArr.SetValue(typeof(int),0);
            myTypeArr.SetValue(typeof(int),1);
            // Get the PropertyInfo object for the indexed property Item,
which has two integer parameters.
            PropertyInfo myPropInfo = myType.GetProperty("Item",
myTypeArr);
            // Display the property.
            Console.WriteLine("The {0} property exists in MyClass3.",
myPropInfo.ToString());
        }
        catch(NullReferenceException e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Source : {0}" , e.Source);
            Console.WriteLine("Message : {0}" , e.Message);
        }
    }
}
```

Remarks

A property is considered public to reflection if it has at least one accessor that is public. Otherwise the property is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

The search for `name` is case-sensitive. The search includes public static and public instance properties.

If the current Type represents a constructed generic type, this method returns the PropertyInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo
- String
- DefaultBinder
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String, Type, Type[])

Searches for the specified public property whose parameters match the specified argument types.

C#

```
public System.Reflection.PropertyInfo? GetProperty (string name, Type?
returnType, Type[] types);
```

Parameters

`name` String

The string containing the name of the public property to get.

`returnType` Type

The return type of the property.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the indexed property to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a property that is not indexed.

Returns

PropertyInfo

An object representing the public property whose parameters match the specified argument types, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified argument types.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

ArgumentException

`types` is multidimensional.

NullReferenceException

An element of `types` is `null`.

Remarks

A property is considered public to reflection if it has at least one accessor that is public. Otherwise the property is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

The search for `name` is case-sensitive. The search includes public static and public instance properties.

If the current Type represents a constructed generic type, this method returns the PropertyInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo
- String
- DefaultBinder
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String, BindingFlags)

Searches for the specified property, using the specified binding constraints.

C#

```
public System.Reflection.PropertyInfo? GetProperty (string name,
System.Reflection.BindingFlags bindingAttr);
```

Parameters

`name` String

The string containing the name of the property to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

Returns

PropertyInfo

An object representing the property that matches the specified requirements, if found; otherwise, `null`.

Implements

GetProperty(String, BindingFlags)

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

Examples

The following example retrieves the type of a user-defined class, retrieves a property of that class and displays the property name in accordance with the specified binding constraints.

C#

```
using System;
using System.Reflection;

class MyClass2
{
```

```
    private int myProperty;
    // Declare MyProperty.
    public int MyProperty
    {
        get
        {
            return myProperty;
        }
        set
        {
            myProperty=value;
        }
    }
}

public class MyTypeClass2
{
    public static void Main(string[] args)
    {
        try
        {
            // Get Type object of MyClass2.
            Type myType=typeof(MyClass2);
            // Get the PropertyInfo by passing the property name and
specifying the BindingFlags.
            PropertyInfo myPropInfo = myType.GetProperty("MyProperty",
BindingFlags.Public | BindingFlags.Instance);
            // Display Name property to console.
            Console.WriteLine("{0} is a property of MyClass2.",
myPropInfo.Name);
        }
        catch(NullReferenceException e)
        {
            Console.WriteLine("MyProperty does not exist in MyClass2."
+e.Message);
        }
    }
}
```

See also

- PropertyInfo
- String
- BindingFlags
- DefaultBinder
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String)

Searches for the public property with the specified name.

```
C#

public System.Reflection.PropertyInfo? GetProperty (string name);
```

Parameters

`name` String

The string containing the name of the public property to get.

Returns

PropertyInfo

An object representing the public property with the specified name, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name.

ArgumentNullException

`name` is `null`.

Examples

The following example retrieves the `Type` object of a user-defined class, retrieves a property of that class, and displays the property name.

C#

```
using System;
using System.Reflection;

class MyClass1
{
    private int myProperty;
    // Declare MyProperty.
    public int MyProperty
    {
        get
        {
            return myProperty;
        }
        set
        {
            myProperty=value;
        }
    }
}

public class MyTypeClass2
{
    public static void Main(string[] args)
    {
        try
        {
            // Get the Type object corresponding to MyClass1.
            Type myType=typeof(MyClass1);
            // Get the PropertyInfo object by passing the property name.
            PropertyInfo myPropInfo = myType.GetProperty("MyProperty");
            // Display the property name.
            Console.WriteLine("The {0} property exists in MyClass1.",
myPropInfo.Name);
        }
        catch(NullReferenceException e)
        {
            Console.WriteLine("The property does not exist in MyClass1."
+ e.Message);
        }
    }
}
```

Remarks

For more information about this API, see Supplemental API remarks for Type.GetProperty.

See also

- PropertyInfo
- String
- DefaultBinder
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetProperty(String, Type)

Searches for the public property with the specified name and return type.

C#

```
public System.Reflection.PropertyInfo? GetProperty (string name, Type?
returnType);
```

Parameters

`name` String

The string containing the name of the public property to get.

`returnType` Type

The return type of the property.

Returns

PropertyInfo

An object representing the public property with the specified name, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name.

ArgumentNullException

`name` is `null`, or `returnType` is `null`.

Examples

The following example defines a class with one property and retrieves the name and type of the property.

C#

```
using System;
using System.Reflection;

class MyPropertyTypeClass
{
    String myMessage = "Hello World.";
    public string MyProperty1
    {
        get
        {
            return myMessage;
        }
        set
        {
            myMessage = value;
        }
    }
}

class TestClass
{
    static void Main()
    {
        try
        {
            Type myType = typeof(MyPropertyTypeClass);
```

```
            // Get the PropertyInfo object representing MyProperty1.
            PropertyInfo myStringProperties1 =
myType.GetProperty("MyProperty1", typeof(string));
            Console.WriteLine("The name of the first property of
MyPropertyTypeClass is {0}.", myStringProperties1.Name);
            Console.WriteLine("The type of the first property of
MyPropertyTypeClass is {0}.", myStringProperties1.PropertyType);
        }
        catch(ArgumentNullException e)
        {
            Console.WriteLine("ArgumentNullException :" + e.Message);
        }
        catch(AmbiguousMatchException e)
        {
            Console.WriteLine("AmbiguousMatchException :" + e.Message);
        }
        catch(NullReferenceException e)
        {
            Console.WriteLine("Source : {0}" , e.Source);
            Console.WriteLine("Message : {0}" , e.Message);
        }
    //Output:
    //The name of the first property of MyPropertyTypeClass is
MyProperty1.
    //The type of the first property of MyPropertyTypeClass is
System.String.
    }
}
```

Remarks

A property is considered public to reflection if it has at least one accessor that is public. Otherwise the property is considered private, and you must use BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.Static (in Visual Basic, combine the values using `Or`) to get it.

The search for `name` is case-sensitive. The search includes public static and public instance properties.

If the current Type represents a constructed generic type, this method returns the PropertyInfo with the type parameters replaced by the appropriate type arguments.

If the current Type represents a type parameter in the definition of a generic type or generic method, this method searches the properties of the class constraint.

See also

- PropertyInfo

- String
- DefaultBinder
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetPropertyImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.

C#

```
protected abstract System.Reflection.PropertyInfo? GetPropertyImpl (string
name, System.Reflection.BindingFlags bindingAttr, System.Reflection.Binder?
binder, Type? returnType, Type[]? types,
System.Reflection.ParameterModifier[]? modifiers);
```

Parameters

`name` String

The string containing the name of the property to get.

`bindingAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted.

-or-

Default to return `null`.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded member, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder.

`returnType` Type

The return type of the property.

`types` Type[]

An array of Type objects representing the number, order, and type of the parameters for the indexed property to get.

-or-

An empty array of the type Type (that is, Type[] types = new Type[0]) to get a property that is not indexed.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `types` array. The default binder does not process this parameter.

Returns

PropertyInfo

An object representing the property that matches the specified requirements, if found; otherwise, `null`.

Exceptions

AmbiguousMatchException

More than one property is found with the specified name and matching the specified binding constraints.

ArgumentNullException

`name` is `null`.

-or-

`types` is `null`.

-or-

One of the elements in `types` is `null`.

ArgumentException

`types` is multidimensional.

-or-

`modifiers` is multidimensional.

-or-

`types` and `modifiers` do not have the same length.

NotSupportedException
The current type is a TypeBuilder, EnumBuilder, or GenericTypeParameterBuilder.

Remarks

Although the default binder does not process ParameterModifier (the `modifiers` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

The following BindingFlags filter flags can be used to define which properties to include in the search:

- You must specify either `BindingFlags.Instance` or `BindingFlags.Static` in order to get a return.
- Specify `BindingFlags.Public` to include public properties in the search.
- Specify `BindingFlags.NonPublic` to include non-public properties (that is, private, internal, and protected properties) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include `public` and `protected` static members up the hierarchy; `private` static members in inherited classes are not included.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the properties declared on the Type, not properties that were simply inherited.

See System.Reflection.BindingFlags for more information.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- PropertyInfo
- String
- BindingFlags
- Binder
- DefaultBinder
- ParameterModifier
- GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GetProperties(BindingFlags)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetRequiredCustomModifiers Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, returns the required custom modifiers of the current Type.

C#

```
public virtual Type[] GetRequiredCustomModifiers ();
```

Returns

Type[]

An array of Type objects that identify the required custom modifiers of the current Type.

-or-

An empty array of type Type, if the current Type has no custom modifiers.

-or-

An empty array of type Type, if the current Type is not a modified Type. A modified Type is obtained from GetModifiedFieldType(), GetModifiedPropertyType(), or GetModifiedParameterType().

Remarks

The GetRequiredCustomModifiers and GetRequiredCustomModifiers methods are provided for designers of managed compilers. For more information on custom modifiers, see classes in the System.Runtime.CompilerServices namespace. Also see the metadata specification in Partition II of the Common Language Infrastructure (CLI) documentation.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a Type object that represents the specified type.

Overloads

Expand table

GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean)	Gets the type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
GetType()	Gets the current Type.
GetType(String)	Gets the Type with the specified name, performing a case-sensitive search.
GetType(String, Boolean)	Gets the Type with the specified name, performing a case-sensitive search and specifying whether to throw an exception if the type is not found.
GetType(String, Boolean, Boolean)	Gets the Type with the specified name, specifying whether to throw an exception if the type is not found and whether to perform a case-sensitive search.
GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>)	Gets the type with the specified name, optionally providing custom methods to resolve the assembly and the type.
GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean)	Gets the type with the specified name, specifying whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.

GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean)

Gets the type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.

C#

```
public static Type? GetType (string typeName,
Func<System.Reflection.AssemblyName,System.Reflection.Assembly?>?
assemblyResolver, Func<System.Reflection.Assembly?,string,bool,Type?>?
typeResolver, bool throwOnError, bool ignoreCase);
```

Parameters

`typeName` String

The name of the type to get. If the `typeResolver` parameter is provided, the type name can be any string that `typeResolver` is capable of resolving. If the `assemblyResolver` parameter is provided or if standard type resolution is used, `typeName` must be an assembly-qualified name (see AssemblyQualifiedName), unless the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, in which case it is sufficient to supply the type name qualified by its namespace.

`assemblyResolver` Func<AssemblyName,Assembly>

A method that locates and returns the assembly that is specified in `typeName`. The assembly name is passed to `assemblyResolver` as an AssemblyName object. If `typeName` does not contain the name of an assembly, `assemblyResolver` is not called. If `assemblyResolver` is not supplied, standard assembly resolution is performed.

Caution Do not pass methods from unknown or untrusted callers. Doing so could result in elevation of privilege for malicious code. Use only methods that you provide or that you are familiar with.

`typeResolver` Func<Assembly,String,Boolean,Type>

A method that locates and returns the type that is specified by `typeName` from the assembly that is returned by `assemblyResolver` or by standard assembly resolution. If no assembly is provided, the method can provide one. The method also takes a parameter that specifies whether to perform a case-insensitive search; the value of `ignoreCase` is passed to that parameter.

Caution Do not pass methods from unknown or untrusted callers.

`throwOnError` Boolean

`true` to throw an exception if the type cannot be found; `false` to return `null`. Specifying `false` also suppresses some other exception conditions, but not all of them. See the Exceptions section.

`ignoreCase` Boolean

`true` to perform a case-insensitive search for `typeName`, `false` to perform a case-sensitive search for `typeName`.

Returns

Type

The type with the specified name. If the type is not found, the `throwOnError` parameter specifies whether `null` is returned or an exception is thrown. In some cases, an exception is thrown regardless of the value of `throwOnError`. See the Exceptions section.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

TypeLoadException

`throwOnError` is `true` and the type is not found.

-or-

`throwOnError` is `true` and `typeName` contains invalid characters, such as an embedded tab.

-or-

`throwOnError` is `true` and `typeName` is an empty string.

-or-

`throwOnError` is `true` and `typeName` represents an array type with an invalid size.

-or-

`typeName` represents an array of TypedReference.

ArgumentException

An error occurs when `typeName` is parsed into a type name and an assembly name (for example, when the simple type name includes an unescaped special character).

-or-

`throwOnError` is `true` and `typeName` contains invalid syntax (for example, "MyType[,*,]").

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

FileNotFoundException

`throwOnError` is `true` and the assembly or one of its dependencies was not found.

FileLoadException

The assembly or one of its dependencies was found, but could not be loaded.

-or-

`typeName` contains an invalid assembly name.

-or-

`typeName` is a valid assembly name without a type name.

BadImageFormatException
The assembly or one of its dependencies is not valid.

-or-

The assembly was compiled with a later version of the common language runtime than the version that is currently loaded.

Remarks

For more information about this API, see Supplemental API remarks for Type.GetType.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetType()

Gets the current Type.

C#

```
public Type GetType ();
```

Returns

Type

The current Type.

Exceptions

TargetInvocationException

A class initializer is invoked and throws an exception.

See also

- TypeLoadException
- Specifying Fully Qualified Type Names

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetType(String)

Gets the Type with the specified name, performing a case-sensitive search.

C#

```
public static Type? GetType (string typeName);
```

Parameters

`typeName` String

The assembly-qualified name of the type to get. See AssemblyQualifiedName. If the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, it is sufficient to supply the type name qualified by its namespace.

Returns

Type

The type with the specified name, if found; otherwise, `null`.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

ArgumentException

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

TypeLoadException

`typeName` represents an array of TypedReference.

FileLoadException

The assembly or one of its dependencies was found, but could not be loaded.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, IOException, instead.

BadImageFormatException

The assembly or one of its dependencies is not valid.

-or-

Version 2.0 or later of the common language runtime is currently loaded, and the assembly was compiled with a later version.

Examples

The following example retrieves the type of `System.Int32` and uses that type object to display the FullName property of `System.Int32`.

C#

```
using System;

class Example
{
     public static void Main()
     {
         try {
             // Get the type of a specified class.
             Type myType1 = Type.GetType("System.Int32");
             Console.WriteLine("The full name is {0}.\n",
myType1.FullName);
         }
         catch (TypeLoadException e)
         {
            Console.WriteLine("{0}: Unable to load type System.Int32",
e.GetType().Name);
         }

         try {
             // Since NoneSuch does not exist in this assembly, GetType
throws a TypeLoadException.
             Type myType2 = Type.GetType("NoneSuch", true);
             Console.WriteLine("The full name is {0}.",
myType2.FullName);
         }
         catch(TypeLoadException e) {
            Console.WriteLine("{0}: Unable to load type NoneSuch",
e.GetType().Name);
         }
     }
}
// The example displays the following output:
//       The full name is System.Int32.
//
//       TypeLoadException: Unable to load type NoneSuch
```

Remarks

You can use the GetType method to obtain a Type object for a type in another assembly if you know its assembly-qualified name, which can be obtained from AssemblyQualifiedName. GetType causes loading of the assembly specified in `typeName`. You can also load an assembly using the Assembly.Load method, and then use the Assembly.GetType or Assembly.GetTypes method to get Type objects. If a type is in an assembly known to your program at compile time, it is more efficient to use `typeof` in C# or the `GetType` operator in Visual Basic.

ⓘ **Note**

If `typeName` cannot be found, the call to the **GetType(String)** method returns `null`. It does not throw an exception. To control whether an exception is thrown, call an overload of the **GetType** method that has a `throwOnError` parameter.

GetType only works on assemblies loaded from disk. If you call GetType to look up a type defined in a dynamic assembly defined using the System.Reflection.Emit services, you might get inconsistent behavior. The behavior depends on whether the dynamic assembly is persistent, that is, created using the `RunAndSave` or `Save` access modes of the System.Reflection.Emit.AssemblyBuilderAccess enumeration. If the dynamic assembly is persistent and has been written to disk before `GetType` is called, the loader finds the saved assembly on disk, loads that assembly, and retrieves the type from that assembly. If the assembly has not been saved to disk when `GetType` is called, the method returns `null`. `GetType` does not understand transient dynamic assemblies; therefore, calling `GetType` to retrieve a type in a transient dynamic assembly returns `null`.

To use `GetType` on a dynamic module, subscribe to the AppDomain.AssemblyResolve event and call `GetType` before saving. Otherwise, you will get two copies of the assembly in memory.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection

Member Type	Static	Non-Static
	Treat properties as hide-by-name-and-signature. See note 2 below.	

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

Arrays or COM types are not searched for unless they have already been loaded into the table of available classes.

`typeName` can be the type name qualified by its namespace or an assembly-qualified name that includes an assembly name specification. See AssemblyQualifiedName.

If `typeName` includes the namespace but not the assembly name, this method searches only the calling object's assembly and mscorlib.dll/System.Private.CoreLib.dll, in that order. If typeName is fully qualified with the partial or complete assembly name, this method searches in the specified assembly. If the assembly has a strong name, a complete assembly name is required.

The AssemblyQualifiedName property returns a fully qualified type name including nested types, the assembly name, and generic type arguments. All compilers that support the common language runtime will emit the simple name of a nested class, and reflection constructs a mangled name when queried, in accordance with the following conventions.

> **Note**
>
> In the .NET Framework version 2.0, processor architecture is added to assembly identity, and can be specified as part of assembly name strings. For example, "ProcessorArchitecture=msil". However, it is not included in the string returned by the **AssemblyQualifiedName** property, for compatibility reasons. You can also load types by creating an **AssemblyName** object and passing it to an appropriate overload of the **Load** method. You can then use the **Assembly.GetType** method to load types from the assembly. See also **AssemblyName.ProcessorArchitecture**.

Expand table

Delimiter	Meaning
Backslash (\)	Escape character.
Backtick (`)	Precedes one or more digits representing the number of type parameters, located at the end of the name of a generic type.
Brackets ([])	Enclose a generic type argument list, for a constructed generic type; within a type argument list, enclose an assembly-qualified type.
Comma (,)	Precedes the Assembly name.
Period (.)	Denotes namespace identifiers.
Plus sign (+)	Precedes a nested class.

For example, the fully qualified name for a class might look like this:

```
TopNamespace.SubNameSpace.ContainingClass+NestedClass,MyAssembly
```

If the namespace were TopNamespace.Sub+Namespace, then the string would have to precede the plus sign (+) with an escape character (\) to prevent it from being interpreted as a nesting separator. Reflection emits this string as follows:

```
TopNamespace.Sub\+Namespace.ContainingClass+NestedClass,MyAssembly
```

A "++" becomes "\+\+", and a "\" becomes "\\".

This qualified name can be persisted and later used to load the Type. To search for and load a Type, use GetType either with the type name only or with the assembly qualified type name. GetType with the type name only will look for the Type in the caller's assembly and then in the System assembly. GetType with the assembly qualified type name will look for the Type in any assembly.

Type names may include trailing characters that denote additional information about the type, such as whether the type is a reference type, a pointer type or an array type. To retrieve the type name without these trailing characters, use `t.GetElementType().ToString()`, where `t` is the type.

Spaces are relevant in all type name components except the assembly name. In the assembly name, spaces before the ',' separator are relevant, but spaces after the ',' separator are ignored.

The name of a generic type ends with a backtick (`) followed by digits representing the number of generic type arguments. The purpose of this name mangling is to allow compilers to support generic types with the same name but with different numbers of type parameters, occurring in the same scope. For example, reflection returns the mangled names `Tuple`1` and `Tuple`2` from the generic methods `Tuple(Of T)` and `Tuple(Of T0, T1)` in Visual Basic, or `Tuple<T>` and `Tuple<T0, T1>` in Visual C#.

For generic types, the type argument list is enclosed in brackets, and the type arguments are separated by commas. For example, a generic Dictionary<TKey,TValue> has two type parameters. A Dictionary<TKey,TValue> of `MyType` with keys of type String might be represented as follows:

```
System.Collections.Generic.Dictionary`2[System.String,MyType]
```

To specify an assembly-qualified type within a type argument list, enclose the assembly-qualified type within brackets. Otherwise, the commas that separate the parts of the assembly-qualified name are interpreted as delimiting additional type arguments. For example, a Dictionary<TKey,TValue> of `MyType` fromMyAssembly.dll, with keys of type String, might be specified as follows:

```
Type.GetType("System.Collections.Generic.Dictionary`2[System.String,
[MyType,MyAssembly]]")
```

> **Note**
>
> An assembly-qualified type can be enclosed in brackets only when it appears within a type parameter list. The rules for searching assemblies for qualified and unqualified types in type parameter lists are the same as the rules for qualified and unqualified nongeneric types.

Nullable types are a special case of generic types. For example, a nullable Int32 is represented by the string "System.Nullable`1[System.Int32]".

> **Note**
>
> In C#, C++, and Visual Basic you can also get nullable types using type operators. For example, the nullable **Boolean** type is returned by `typeof(Nullable<bool>)` in C#, by `Nullable<Boolean>::typeid` in C++, and by `GetType(Nullable(Of Boolean))` in Visual Basic.

The following table shows the syntax you use with `GetType` for various types.

Expand table

To Get	Use
A nullable Int32	`Type.GetType("System.Nullable`1[System.Int32]")`
An unmanaged pointer to `MyType`	`Type.GetType("MyType*")`
An unmanaged pointer to a pointer to `MyType`	`Type.GetType("MyType**")`
A managed pointer or reference to `MyType`	`Type.GetType("MyType&")`. Note that unlike pointers, references are limited to one level.
A parent class and a nested class	`Type.GetType("MyParentClass+MyNestedClass")`
A one-dimensional array with a lower bound of 0	`Type.GetType("MyType[]")`
A one-dimensional array with an unknown	`Type.GetType("MyType[*]")`

To Get	Use
lower bound	
An n-dimensional array	A comma (,) inside the brackets a total of n-1 times. For example, `System.Object[,,]` represents a three-dimensional `Object` array.
An array of one-dimensional arrays	`Type.GetType("MyType[][]")`
A rectangular two-dimensional array with unknown lower bounds	`Type.GetType("MyType[,]")`
A generic type with one type argument	`Type.GetType("MyGenericType`1[MyType]")`
A generic type with two type arguments	`Type.GetType("MyGenericType`2[MyType,AnotherType]")`
A generic type with two assembly-qualified type arguments	`Type.GetType("MyGenericType`2[[MyType,MyAssembly],[AnotherType,AnotherAssembly]]")`
An assembly-qualified generic type with an assembly-qualified type argument	`Type.GetType("MyGenericType`1[[MyType,MyAssembly]],MyGenericTypeAssembly")`

To Get	Use
A generic type whose type argument is a generic type with two type arguments	`Type.GetType("MyGenericType`1[AnotherGenericType`2[MyType,AnotherType]]")`

See also

- String
- TypeLoadException
- AssemblyQualifiedName
- GetAssembly(Type)
- GetType(String)
- AssemblyName
- Specifying Fully Qualified Type Names

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetType(String, Boolean)

Gets the Type with the specified name, performing a case-sensitive search and specifying whether to throw an exception if the type is not found.

```
C#
```

```
public static Type? GetType (string typeName, bool throwOnError);
```

Parameters

`typeName` String

The assembly-qualified name of the type to get. See AssemblyQualifiedName. If the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, it is sufficient to supply the type name qualified by its namespace.

`throwOnError` Boolean

`true` to throw an exception if the type cannot be found; `false` to return `null`. Specifying `false` also suppresses some other exception conditions, but not all of them. See the Exceptions section.

Returns

Type

The type with the specified name. If the type is not found, the `throwOnError` parameter specifies whether `null` is returned or an exception is thrown. In some cases, an exception is thrown regardless of the value of `throwOnError`. See the Exceptions section.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

TypeLoadException

`throwOnError` is `true` and the type is not found.

-or-

`throwOnError` is `true` and `typeName` contains invalid characters, such as an embedded tab.

-or-

`throwOnError` is `true` and `typeName` is an empty string.

-or-

`throwOnError` is `true` and `typeName` represents an array type with an invalid size.

-or-

`typeName` represents an array of TypedReference.

ArgumentException

`throwOnError` is `true` and `typeName` contains invalid syntax. For example, "MyType[,*,]".

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

FileNotFoundException

`throwOnError` is `true` and the assembly or one of its dependencies was not found.

FileLoadException

The assembly or one of its dependencies was found, but could not be loaded.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, IOException, instead.

BadImageFormatException

The assembly or one of its dependencies is not valid.

-or-

Version 2.0 or later of the common language runtime is currently loaded, and the assembly was compiled with a later version.

Examples

The following example retrieves the type of `System.Int32` and uses that type object to display the FullName property of `System.Int32`. If a type object refers to an assembly that does not exist, this example throws an exception.

C#

```
using System;

class Example
{
     public static void Main()
     {
         try {
             // Get the type of a specified class.
             Type myType1 = Type.GetType("System.Int32");
             Console.WriteLine("The full name is {0}.\n",
myType1.FullName);
         }
         catch (TypeLoadException e)
         {
            Console.WriteLine("{0}: Unable to load type System.Int32",
e.GetType().Name);
         }

         try {
             // Since NoneSuch does not exist in this assembly, GetType
throws a TypeLoadException.
             Type myType2 = Type.GetType("NoneSuch", true);
             Console.WriteLine("The full name is {0}.",
myType2.FullName);
         }
         catch(TypeLoadException e) {
            Console.WriteLine("{0}: Unable to load type NoneSuch",
e.GetType().Name);
         }
     }
}
// The example displays the following output:
//       The full name is System.Int32.
//
//       TypeLoadException: Unable to load type NoneSuch
```

Remarks

You can use the GetType method to obtain a Type object for a type in another assembly if you know its assembly-qualified name, which can be obtained from AssemblyQualifiedName. GetType causes loading of the assembly specified in

`typeName`. You can also load an assembly using the Assembly.Load method, and then use the Assembly.GetType or Assembly.GetTypes method to get Type objects. If a type is in an assembly known to your program at compile time, it is more efficient to use `typeof` in C# or the `GetType` operator in Visual Basic.

`GetType` only works on assemblies loaded from disk. If you call `GetType` to look up a type defined in a dynamic assembly defined using the System.Reflection.Emit services, you might get inconsistent behavior. The behavior depends on whether the dynamic assembly is persistent, that is, created using the `RunAndSave` or `Save` access modes of the System.Reflection.Emit.AssemblyBuilderAccess enumeration. If the dynamic assembly is persistent and has been written to disk before `GetType` is called, the loader finds the saved assembly on disk, loads that assembly, and retrieves the type from that assembly. If the assembly has not been saved to disk when `GetType` is called, the method returns `null`. `GetType` does not understand transient dynamic assemblies; therefore, calling `GetType` to retrieve a type in a transient dynamic assembly returns `null`.

To use `GetType` on a dynamic module, subscribe to the AppDomain.AssemblyResolve event and call `GetType` before saving. Otherwise, you will get two copies of the assembly in memory.

The `throwOnError` parameter specifies what happens when the type is not found, and also suppresses certain other exception conditions, as described in the Exceptions section. Some exceptions are thrown regardless of the value of `throwOnError`. For example, if the type is found but cannot be loaded, a TypeLoadException is thrown even if `throwOnError` is `false`.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

Member Type	Static	Non-Static
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

Arrays or COM types are not searched for unless they have already been loaded into the table of available classes.

`typeName` can be the type name qualified by its namespace or an assembly-qualified name that includes an assembly name specification. See AssemblyQualifiedName.

If `typeName` includes the namespace but not the assembly name, this method searches only the calling object's assembly and mscorlib.dll/System.Private.CoreLib.dll, in that order. If typeName is fully qualified with the partial or complete assembly name, this method searches in the specified assembly. If the assembly has a strong name, a complete assembly name is required.

The AssemblyQualifiedName property returns a fully qualified type name including nested types, the assembly name, and generic arguments. All compilers that support the common language runtime will emit the simple name of a nested class, and reflection constructs a mangled name when queried, in accordance with the following conventions.

Note

In the .NET Framework version 2.0, processor architecture is added to assembly identity, and can be specified as part of assembly name strings. For example, "ProcessorArchitecture=msil". However, it is not included in the string returned by the **AssemblyQualifiedName** property, for compatibility reasons. You can also load types by creating an **AssemblyName** object and passing it to an appropriate overload of the **Load** method. You can then use the **Assembly.GetType** method to load types from the assembly. See also **AssemblyName.ProcessorArchitecture**.

Expand table

Delimiter	Meaning
Backslash (\)	Escape character.
Backtick (`)	Precedes one or more digits representing the number of type parameters, located at the end of the name of a generic type.
Brackets ([])	Enclose a generic type argument list, for a constructed generic type; within a type argument list, enclose an assembly-qualified type.
Comma (,)	Precedes the Assembly name.
Period (.)	Denotes namespace identifiers.
Plus sign (+)	Precedes a nested class.

For example, the fully qualified name for a class might look like this:

```
TopNamespace.SubNameSpace.ContainingClass+NestedClass,MyAssembly
```

If the namespace were TopNamespace.Sub+Namespace, then the string would have to precede the plus sign (+) with an escape character (\) to prevent it from being interpreted as a nesting separator. Reflection emits this string as follows:

```
TopNamespace.Sub\+Namespace.ContainingClass+NestedClass,MyAssembly
```

A "++" becomes "\+\+", and a "\" becomes "\\".

This qualified name can be persisted and later used to load the Type. To search for and load a Type, use GetType either with the type name only or with the assembly qualified type name. GetType with the type name only will look for the Type in the caller's assembly and then in the System assembly. GetType with the assembly qualified type name will look for the Type in any assembly.

Type names may include trailing characters that denote additional information about the type, such as whether the type is a reference type, a pointer type or an array type. To retrieve the type name without these trailing characters, use `t.GetElementType().ToString()`, where `t` is the type.

Spaces are relevant in all type name components except the assembly name. In the assembly name, spaces before the ',' separator are relevant, but spaces after the ',' separator are ignored.

The name of a generic type ends with a backtick (`) followed by digits representing the number of generic type arguments. The purpose of this name mangling is to allow compilers to support generic types with the same name but with different numbers of type parameters, occurring in the same scope. For example, reflection returns the mangled names `Tuple`1` and `Tuple`2` from the generic methods `Tuple(Of T)` and `Tuple(Of T0, T1)` in Visual Basic, or `Tuple<T>` and `Tuple<T0, T1>` in Visual C#.

For generic types, the type argument list is enclosed in brackets, and the type arguments are separated by commas. For example, a generic Dictionary<TKey,TValue> has two type parameters. A Dictionary<TKey,TValue> of `MyType` with keys of type String might be represented as follows:

```
System.Collections.Generic.Dictionary`2[System.String,MyType]
```

To specify an assembly-qualified type within a type argument list, enclose the assembly-qualified type within brackets. Otherwise, the commas that separate the parts of the assembly-qualified name are interpreted as delimiting additional type arguments. For example, a Dictionary<TKey,TValue> of `MyType` from MyAssembly.dll, with keys of type String, might be specified as follows:

```
Type.GetType("System.Collections.Generic.Dictionary`2[System.String,
[MyType,MyAssembly]]")
```

> **Note**
>
> An assembly-qualified type can be enclosed in brackets only when it appears within a type parameter list. The rules for searching assemblies for qualified and unqualified types in type parameter lists are the same as the rules for qualified and unqualified nongeneric types.

Nullable types are a special case of generic types. For example, a nullable Int32 is represented by the string "System.Nullable`1[System.Int32]".

> **Note**
>
> In C#, C++, and Visual Basic you can also get nullable types using type operators. For example, the nullable **Boolean** type is returned by `typeof(Nullable<bool>)` in C#, by `Nullable<Boolean>::typeid` in C++, and by `GetType(Nullable(Of Boolean))` in Visual Basic.

The following table shows the syntax you use with `GetType` for various types.

Expand table

To Get	Use
A nullable Int32	`Type.GetType("System.Nullable`1[System.Int32]")`
An unmanaged pointer to `MyType`	`Type.GetType("MyType*")`
An unmanaged pointer to a pointer to `MyType`	`Type.GetType("MyType**")`
A managed pointer or reference to `MyType`	`Type.GetType("MyType&")`. Note that unlike pointers, references are limited to one level.
A parent class and a	`Type.GetType("MyParentClass+MyNestedClass")`

To Get	Use
nested class	
A one-dimensional array with a lower bound of 0	`Type.GetType("MyArray[]")`
A one-dimensional array with an unknown lower bound	`Type.GetType("MyArray[*]")`
An n-dimensional array	A comma (,) inside the brackets a total of n-1 times. For example, `System.Object[,,]` represents a three-dimensional `Object` array.
A two-dimensional array's array	`Type.GetType("MyArray[][]")`
A rectangular two-dimensional array with unknown lower bounds	`Type.GetType("MyArray[,]")`
A generic type with one type argument	`Type.GetType("MyGenericType`1[MyType]")`
A generic type with two type arguments	`Type.GetType("MyGenericType`2[MyType,AnotherType]")`
A generic type with two assembly-qualified type arguments	`Type.GetType("MyGenericType`2[[MyType,MyAssembly],[AnotherType,AnotherAssembly]]")`

To Get	Use
An assembly-qualified generic type with an assembly-qualified type argument	`Type.GetType("MyGenericType`1[[MyType,MyAssembly]],MyGenericTypeAssembly")`
A generic type whose type argument is a generic type with two type arguments	`Type.GetType("MyGenericType`1[AnotherGenericType`2[MyType,AnotherType]]")`

See also

- String
- TypeLoadException
- AssemblyQualifiedName
- GetAssembly(Type)
- GetType(String)
- AssemblyName
- Specifying Fully Qualified Type Names

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetType(String, Boolean, Boolean)

Gets the Type with the specified name, specifying whether to throw an exception if the type is not found and whether to perform a case-sensitive search.

```
C#

public static Type? GetType (string typeName, bool throwOnError, bool
ignoreCase);
```

Parameters

`typeName` String

The assembly-qualified name of the type to get. See AssemblyQualifiedName. If the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, it is sufficient to supply the type name qualified by its namespace.

`throwOnError` Boolean

`true` to throw an exception if the type cannot be found; `false` to return `null`. Specifying `false` also suppresses some other exception conditions, but not all of them. See the Exceptions section.

`ignoreCase` Boolean

`true` to perform a case-insensitive search for `typeName`, `false` to perform a case-sensitive search for `typeName`.

Returns

Type

The type with the specified name. If the type is not found, the `throwOnError` parameter specifies whether `null` is returned or an exception is thrown. In some cases, an exception is thrown regardless of the value of `throwOnError`. See the Exceptions section.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

TypeLoadException

`throwOnError` is `true` and the type is not found.

-or-

`throwOnError` is `true` and `typeName` contains invalid characters, such as an embedded tab.

-or-

`throwOnError` is `true` and `typeName` is an empty string.

-or-

`throwOnError` is `true` and `typeName` represents an array type with an invalid size.

-or-

`typeName` represents an array of TypedReference.

ArgumentException

`throwOnError` is `true` and `typeName` contains invalid syntax. For example, "MyType[,*,]".

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

FileNotFoundException

`throwOnError` is `true` and the assembly or one of its dependencies was not found.

FileLoadException

The assembly or one of its dependencies was found, but could not be loaded.

BadImageFormatException
The assembly or one of its dependencies is not valid.

-or-

Version 2.0 or later of the common language runtime is currently loaded, and the assembly was compiled with a later version.

Remarks

You can use the GetType method to obtain a Type object for a type in another assembly if you know its assembly-qualified name, which can be obtained from AssemblyQualifiedName. GetType causes loading of the assembly specified in `typeName`. You can also load an assembly using the Assembly.Load method, and then use the Assembly.GetType or Assembly.GetTypes method to get Type objects. If a type is in an assembly known to your program at compile time, it is more efficient to use `typeof` in C# or the `GetType` operator in Visual Basic.

`GetType` only works on assemblies loaded from disk. If you call `GetType` to look up a type defined in a dynamic assembly defined using the System.Reflection.Emit services, you might get inconsistent behavior. The behavior depends on whether the dynamic assembly is persistent, that is, created using the `RunAndSave` or `Save` access modes of the System.Reflection.Emit.AssemblyBuilderAccess enumeration. If the dynamic assembly is persistent and has been written to disk before `GetType` is called, the loader finds the saved assembly on disk, loads that assembly, and retrieves the type from that assembly. If the assembly has not been saved to disk when `GetType` is called, the method returns `null`. `GetType` does not understand transient dynamic assemblies; therefore, calling `GetType` to retrieve a type in a transient dynamic assembly returns `null`.

To use `GetType` on a dynamic module, subscribe to the AppDomain.AssemblyResolve event and call `GetType` before saving. Otherwise, you will get two copies of the assembly in memory.

The `throwOnError` parameter specifies what happens when the type is not found, and also suppresses certain other exception conditions, as described in the Exceptions section. Some exceptions are thrown regardless of the value of `throwOnError`. For example, if the type is found but cannot be loaded, a TypeLoadException is thrown even if `throwOnError` is `false`.

The following table shows what members of a base class are returned by the `Get` methods when reflecting on a type.

Expand table

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the methods that implement the property. Reflection treats properties as hide-by-name-and-signature. See note 2 below.

1. Hide-by-name-and-signature considers all of the parts of the signature, including custom modifiers, return types, parameter types, sentinels, and unmanaged calling conventions. This is a binary comparison.

2. For reflection, properties and events are hide-by-name-and-signature. If you have a property with both a get and a set accessor in the base class, but the derived class has only a get accessor, the derived class property hides the base class property, and you will not be able to access the setter on the base class.

3. Custom attributes are not part of the common type system.

Arrays or COM types are not searched for unless they have already been loaded into the table of available classes.

`typeName` can be the type name qualified by its namespace or an assembly-qualified name that includes an assembly name specification. See AssemblyQualifiedName.

If `typeName` includes the namespace but not the assembly name, this method searches only the calling object's assembly and mscorlib.dll/System.Private.CoreLib.dll, in that order. If typeName is fully qualified

with the partial or complete assembly name, this method searches in the specified assembly. If the assembly has a strong name, a complete assembly name is required.

The AssemblyQualifiedName property returns a fully qualified type name including nested types, the assembly name, and type arguments. All compilers that support the common language runtime will emit the simple name of a nested class, and reflection constructs a mangled name when queried, in accordance with the following conventions.

> **Note**
>
> In the .NET Framework version 2.0, processor architecture is added to assembly identity, and can be specified as part of assembly name strings. For example, "ProcessorArchitecture=msil". However, it is not included in the string returned by the **AssemblyQualifiedName** property, for compatibility reasons. You can also load types by creating an **AssemblyName** object and passing it to an appropriate overload of the **Load** method. You can then use the **Assembly.GetType** method to load types from the assembly. See also **AssemblyName.ProcessorArchitecture**.

Expand table

Delimiter	Meaning
Backslash (\)	Escape character.
Backtick (`)	Precedes one or more digits representing the number of type parameters, located at the end of the name of a generic type.
Brackets ([])	Enclose a generic type argument list, for a constructed generic type; within a type argument list, enclose an assembly-qualified type.
Comma (,)	Precedes the Assembly name.
Period (.)	Denotes namespace identifiers.
Plus sign (+)	Precedes a nested class.

For example, the fully qualified name for a class might look like this:

```
TopNamespace.SubNameSpace.ContainingClass+NestedClass,MyAssembly
```

If the namespace were TopNamespace.Sub+Namespace, then the string would have to precede the plus sign (+) with an escape character (\) to prevent it from being interpreted as a nesting separator. Reflection emits this string as follows:

```

TopNamespace.Sub\+Namespace.ContainingClass+NestedClass,MyAssembly
```

A "++" becomes "\+\+", and a "\" becomes "\\".

This qualified name can be persisted and later used to load the Type. To search for and load a Type, use GetType either with the type name only or with the assembly qualified type name. GetType with the type name only will look for the Type in the caller's assembly and then in the System assembly. GetType with the assembly qualified type name will look for the Type in any assembly.

Type names may include trailing characters that denote additional information about the type, such as whether the type is a reference type, a pointer type or an array type. To retrieve the type name without these trailing characters, use `t.GetElementType().ToString()`, where `t` is the type.

Spaces are relevant in all type name components except the assembly name. In the assembly name, spaces before the ',' separator are relevant, but spaces after the ',' separator are ignored.

The name of a generic type ends with a backtick (`) followed by digits representing the number of generic type arguments. The purpose of this name mangling is to allow compilers to support generic types with the same name but with different numbers of type parameters, occurring in the same scope. For example, reflection returns the mangled names `Tuple`1` and `Tuple`2` from the generic methods `Tuple(Of T)` and `Tuple(Of T0, T1)` in Visual Basic, or `Tuple<T>` and `Tuple<T0, T1>` in Visual C#.

For generic types, the type argument list is enclosed in brackets, and the type arguments are separated by commas. For example, a generic Dictionary<TKey,TValue> has two type parameters. A Dictionary<TKey,TValue> of `MyType` with keys of type String might be represented as follows:

```

```

```
System.Collections.Generic.Dictionary`2[System.String,MyType]
```

To specify an assembly-qualified type within a type argument list, enclose the assembly-qualified type within brackets. Otherwise, the commas that separate the parts of the assembly-qualified name are interpreted as delimiting additional type arguments. For example, a Dictionary<TKey,TValue> of `MyType` from MyAssembly.dll, with keys of type String, might be specified as follows:

```
Type.GetType("System.Collections.Generic.Dictionary`2[System.String,
[MyType,MyAssembly]]")
```

> **Note**
>
> An assembly-qualified type can be enclosed in brackets only when it appears within a type parameter list. The rules for searching assemblies for qualified and unqualified types in type parameter lists are the same as the rules for qualified and unqualified nongeneric types.

Nullable types are a special case of generic types. For example, a nullable Int32 is represented by the string "System.Nullable`1[System.Int32]".

> **Note**
>
> In C#, C++, and Visual Basic you can also get nullable types using type operators. For example, the nullable **Boolean** type is returned by `typeof(Nullable<bool>)` in C#, by `Nullable<Boolean>::typeid` in C++, and by `GetType(Nullable(Of Boolean))` in Visual Basic.

The following table shows the syntax you use with `GetType` for various types.

Expand table

To Get	Use
A nullable Int32	`Type.GetType("System.Nullable`1[System.Int32]")`
An unmanaged	`Type.GetType("MyType*")`

To Get	Use
pointer to `MyType`	
An unmanaged pointer to a pointer to `MyType`	`Type.GetType("MyType**")`
A managed pointer or reference to `MyType`	`Type.GetType("MyType&")`. Note that unlike pointers, references are limited to one level.
A parent class and a nested class	`Type.GetType("MyParentClass+MyNestedClass")`
A one-dimensional array with a lower bound of 0	`Type.GetType("MyArray[]")`
A one-dimensional array with an unknown lower bound	`Type.GetType("MyArray[*]")`
An n-dimensional array	A comma (,) inside the brackets a total of n-1 times. For example, `System.Object[,,]` represents a three-dimensional `Object` array.
A two-dimensional array's array	`Type.GetType("MyArray[][]")`
A rectangular two-dimensional array with unknown lower bounds	`Type.GetType("MyArray[,]")`

To Get	Use
A generic type with one type argument	`Type.GetType("MyGenericType`1[MyType]")`
A generic type with two type arguments	`Type.GetType("MyGenericType`2[MyType,AnotherType]")`
A generic type with two assembly-qualified type arguments	`Type.GetType("MyGenericType`2[[MyType,MyAssembly],[AnotherType,AnotherAssembly]]")`
An assembly-qualified generic type with an assembly-qualified type argument	`Type.GetType("MyGenericType`1[[MyType,MyAssembly]],MyGenericTypeAssembly")`
A generic type whose type argument is a generic type with two type arguments	`Type.GetType("MyGenericType`1[AnotherGenericType`2[MyType,AnotherType]]")`

See also

- String
- TypeLoadException
- AssemblyQualifiedName
- GetAssembly(Type)
- GetType(String)
- AssemblyName

- Specifying Fully Qualified Type Names

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>)

Gets the type with the specified name, optionally providing custom methods to resolve the assembly and the type.

C#

```
public static Type? GetType (string typeName,
Func<System.Reflection.AssemblyName,System.Reflection.Assembly?>?
assemblyResolver, Func<System.Reflection.Assembly?,string,bool,Type?>?
typeResolver);
```

Parameters

`typeName` String

The name of the type to get. If the `typeResolver` parameter is provided, the type name can be any string that `typeResolver` is capable of resolving. If the `assemblyResolver` parameter is provided or if standard type resolution is used, `typeName` must be an assembly-qualified name (see AssemblyQualifiedName), unless the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, in which case it is sufficient to supply the type name qualified by its namespace.

`assemblyResolver` Func<AssemblyName,Assembly>

A method that locates and returns the assembly that is specified in `typeName`. The assembly name is passed to `assemblyResolver` as an AssemblyName object. If `typeName` does not contain the name of an assembly, `assemblyResolver` is not called. If `assemblyResolver` is not supplied, standard assembly resolution is performed.

Caution Do not pass methods from unknown or untrusted callers. Doing so could result in elevation of privilege for malicious code. Use only methods that you provide or that you are familiar with.

`typeResolver` Func<Assembly,String,Boolean,Type>

A method that locates and returns the type that is specified by `typeName` from the assembly that is returned by `assemblyResolver` or by standard assembly resolution. If no assembly is provided, the `typeResolver` method can provide one. The method also takes a parameter that specifies whether to perform a case-insensitive search; `false` is passed to that parameter.

Caution Do not pass methods from unknown or untrusted callers.

Returns

Type

The type with the specified name, or `null` if the type is not found.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

ArgumentException

An error occurs when `typeName` is parsed into a type name and an assembly name (for example, when the simple type name includes an unescaped special character).

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

TypeLoadException
`typeName` represents an array of TypedReference.

FileLoadException
The assembly or one of its dependencies was found, but could not be loaded.

-or-

`typeName` contains an invalid assembly name.

-or-

`typeName` is a valid assembly name without a type name.

BadImageFormatException
The assembly or one of its dependencies is not valid.

-or-

The assembly was compiled with a later version of the common language runtime than the version that is currently loaded.

Remarks

Usage scenarios for this method and details about the `assemblyResolver` and `typeResolver` parameters can be found in the GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean) method overload.

> **Note**
>
> If `typeName` cannot be found, the call to the **GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>)** method returns `null`. It does not throw an exception. To control whether an

exception is thrown, call an overload of the **GetType** method that has a `throwOnError` parameter.

Calling this method overload is the same as calling the GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean) method overload and specifying `false` for the `throwOnError` and `ignoreCase` parameters.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean)

Gets the type with the specified name, specifying whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.

C#

```
public static Type? GetType (string typeName,
Func<System.Reflection.AssemblyName,System.Reflection.Assembly?>?
assemblyResolver, Func<System.Reflection.Assembly?,string,bool,Type?>?
typeResolver, bool throwOnError);
```

Parameters

`typeName` String

The name of the type to get. If the `typeResolver` parameter is provided, the type name can be any string that `typeResolver` is capable of resolving. If the

`assemblyResolver` parameter is provided or if standard type resolution is used, `typeName` must be an assembly-qualified name (see AssemblyQualifiedName), unless the type is in the currently executing assembly or in mscorlib.dll/System.Private.CoreLib.dll, in which case it is sufficient to supply the type name qualified by its namespace.

`assemblyResolver` Func<AssemblyName,Assembly>

A method that locates and returns the assembly that is specified in `typeName`. The assembly name is passed to `assemblyResolver` as an AssemblyName object. If `typeName` does not contain the name of an assembly, `assemblyResolver` is not called. If `assemblyResolver` is not supplied, standard assembly resolution is performed.

Caution Do not pass methods from unknown or untrusted callers. Doing so could result in elevation of privilege for malicious code. Use only methods that you provide or that you are familiar with.

`typeResolver` Func<Assembly,String,Boolean,Type>

A method that locates and returns the type that is specified by `typeName` from the assembly that is returned by `assemblyResolver` or by standard assembly resolution. If no assembly is provided, the method can provide one. The method also takes a parameter that specifies whether to perform a case-insensitive search; `false` is passed to that parameter.

Caution Do not pass methods from unknown or untrusted callers.

`throwOnError` Boolean

`true` to throw an exception if the type cannot be found; `false` to return `null`. Specifying `false` also suppresses some other exception conditions, but not all of them. See the Exceptions section.

Returns

Type

The type with the specified name. If the type is not found, the `throwOnError` parameter specifies whether `null` is returned or an exception is thrown. In some cases, an exception is thrown regardless of the value of `throwOnError`. See the Exceptions section.

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

TypeLoadException

`throwOnError` is `true` and the type is not found.

-or-

`throwOnError` is `true` and `typeName` contains invalid characters, such as an embedded tab.

-or-

`throwOnError` is `true` and `typeName` is an empty string.

-or-

`throwOnError` is `true` and `typeName` represents an array type with an invalid size.

-or-

`typeName` represents an array of TypedReference.

ArgumentException

An error occurs when `typeName` is parsed into a type name and an assembly name (for example, when the simple type name includes an unescaped special character).

-or-

`throwOnError` is `true` and `typeName` contains invalid syntax (for example, "MyType[,*,]").

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

FileNotFoundException
`throwOnError` is `true` and the assembly or one of its dependencies was not found.

-or-

`typeName` contains an invalid assembly name.

-or-

`typeName` is a valid assembly name without a type name.

FileLoadException
The assembly or one of its dependencies was found, but could not be loaded.

BadImageFormatException
The assembly or one of its dependencies is not valid.

-or-

The assembly was compiled with a later version of the common language runtime than the version that is currently loaded.

Remarks

Usage scenarios for this method and details about the `assemblyResolver` and `typeResolver` parameters can be found in the GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean) method overload.

Calling this method overload is the same as calling the GetType(String, Func<AssemblyName,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean) method overload and specifying `false` for the `ignoreCase` parameter.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeArray(Object[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the types of the objects in the specified array.

C#

```
public static Type[] GetTypeArray (object[] args);
```

Parameters

`args` Object[]

An array of objects whose types to determine.

Returns

Type[]

An array of Type objects representing the types of the corresponding elements in `args`.

Exceptions

ArgumentNullException

`args` is `null`.

-or-

One or more of the elements in `args` is `null`.

TargetInvocationException

The class initializers are invoked and at least one throws an exception.

Examples

The following code example demonstrates how to use the GetTypeArray method to list the types of the elements of an array.

C#

```
Object[] myObject = new Object[3];
myObject[0] = 66;
myObject[1] = "puri";
myObject[2] = 33.33;
// Get the array of 'Type' class objects.
Type[] myTypeArray = Type.GetTypeArray(myObject);
Console.WriteLine("Full names of the 'Type' objects in the array are:");
for(int h = 0; h < myTypeArray.Length ; h++)
{
    Console.WriteLine(myTypeArray[h].FullName);
}
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeCode(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the underlying type code of the specified Type.

C#

```
public static TypeCode GetTypeCode (Type? type);
```

Parameters

type Type

The type whose underlying type code to get.

Returns

TypeCode

The code of the underlying type, or Empty if `type` is `null`.

Examples

The following code example demonstrates how the TypeCode enumeration can be used. In a decision block inside the `WriteObjectInfo` method, the TypeCode of an Object parameter is examined, and an appropriate message is written to the console.

C#

```
static void WriteObjectInfo(object testObject)
{
    TypeCode    typeCode = Type.GetTypeCode( testObject.GetType() );

    switch( typeCode )
    {
        case TypeCode.Boolean:
            Console.WriteLine("Boolean: {0}", testObject);
            break;
```

```
        case TypeCode.Double:
            Console.WriteLine("Double: {0}", testObject);
            break;

        default:
            Console.WriteLine("{0}: {1}", typeCode.ToString(), testObject);
            break;
    }
}
```

Remarks

When you inherit from Type, you can change the behavior of this method by overriding the GetTypeCodeImpl method. For Enum types, the type code of the underlying integral type is returned.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.5, 1.6, 2.0, 2.1

See also

- TypeCode
- GetTypeCodeImpl()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeCodeImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the underlying type code of this Type instance.

C#

```
protected virtual TypeCode GetTypeCodeImpl ();
```

Returns

TypeCode

The type code of the underlying type.

Remarks

This method provides the implementation for the `static` (in C#) or `Shared` (in Visual Basic) GetTypeCode(Type) method. When you inherit from Type, you can override this method to provide your own implementation of GetTypeCode.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode
- GetTypeCode(Type)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.GetTypeFromCLSID Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type associated with the specified class identifier (CLSID).

Overloads

Expand table

GetTypeFromCLSID(Guid)	Gets the type associated with the specified class identifier (CLSID).
GetTypeFromCLSID(Guid, Boolean)	Gets the type associated with the specified class identifier (CLSID), specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromCLSID(Guid, String)	Gets the type associated with the specified class identifier (CLSID) from the specified server.
GetTypeFromCLSID(Guid, String, Boolean)	Gets the type associated with the specified class identifier (CLSID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.

GetTypeFromCLSID(Guid)

Gets the type associated with the specified class identifier (CLSID).

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromCLSID (Guid clsid);
```

Parameters

clsid Guid

The CLSID of the type to get.

Returns

Type

`System.__ComObject` regardless of whether the CLSID is valid.

Attributes SupportedOSPlatformAttribute

Examples

The following example uses the CLSID of the Microsoft Word Application object to retrieve a COM type that represents the Microsoft Word application. It then instantiates the type by calling the Activator.CreateInstance method, and closes it by calling the Application.Quit method.

C#

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

public class Example
{
   private const string WORD_CLSID = "{000209FF-0000-0000-C000-
000000000046}";

   public static void Main()
   {
      // Start an instance of the Word application.
      var word = Type.GetTypeFromCLSID(Guid.Parse(WORD_CLSID));
      Console.WriteLine("Instantiated Type object from CLSID {0}",
                        WORD_CLSID);
      Object wordObj = Activator.CreateInstance(word);
      Console.WriteLine("Instantiated {0}",
                        wordObj.GetType().FullName);

      // Close Word.
      word.InvokeMember("Quit", BindingFlags.InvokeMethod, null,
                        wordObj, new object[] { 0, 0, false } );
   }
}
// The example displays the following output:
//    Instantiated Type object from CLSID {000209FF-0000-0000-C000-
000000000046}
//    Instantiated Microsoft.Office.Interop.Word.ApplicationClass
```

Remarks

The GetTypeFromCLSID method supports late-bound access to unmanaged COM objects from .NET Framework apps when you know the COM object's class identifier (CLSID). The class identifier for COM classes is defined in the HKEY_CLASSES_ROOT\CLSID key of the registry. You can retrieve the value of the IsCOMObject property to determine whether the type returned by this method is a COM object.

> **Tip**
>
> You can call the **GetTypeFromProgID** method for late-bound access to COM objects whose programmatic identifier (ProgID) you know.

Instantiating an unmanaged COM object from its CLSID is a two-step process:

1. Get a Type object that represents the `__ComObject` that corresponds to the CLSID by calling the GetTypeFromCLSID method.
2. Call the Activator.CreateInstance(Type) method to instantiate the COM object.

See the example for an illustration.

The GetTypeFromCLSID(Guid) overload ignores any exception that may occur when instantiating a Type object based on the `clsid` argument. Note that no exception is thrown if `clsid` is not found in the registry.

Notes to Callers

This method is intended for use when working with COM objects, not with .NET Framework objects. All managed objects, including those that are visible to COM (that is, their ComVisibleAttribute attribute is `true`) have a GUID that is returned by the GUID property. Although the method returns a Type object that corresponds to the GUID for .NET Framework objects, you can't use that Type object to create a type instance by calling the CreateInstance(Type) method, as the following example shows.

C#

```
using System;
using System.Runtime.InteropServices;

[assembly:ComVisible(true)]

// Define two classes, and assign one an explicit GUID.
[GuidAttribute("d055cba3-1f83-4bd7-ba19-e22b1b8ec3c4")]
```

```
public class ExplicitGuid
{ }

public class NoExplicitGuid
{ }

public class Example
{
   public static void Main()
   {
      Type explicitType = typeof(ExplicitGuid);
      Guid explicitGuid = explicitType.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type explicitCOM = Type.GetTypeFromCLSID(explicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        explicitCOM.Name, explicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        explicitType.Name, explicitCOM.Name,
                        explicitType.Equals(explicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(explicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
      }
      catch (COMException e) {
         Console.WriteLine("COM Exception:\n{0}\n", e.Message);
      }

      Type notExplicit = typeof(NoExplicitGuid);
      Guid notExplicitGuid = notExplicit.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type notExplicitCOM = Type.GetTypeFromCLSID(notExplicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        notExplicitCOM.Name, notExplicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        notExplicit.Name, notExplicitCOM.Name,
                        notExplicit.Equals(notExplicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(notExplicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
      }
      catch (COMException e) {
         Console.WriteLine("COM Exception:\n{0}\n", e.Message);
      }
```

```
    }
}
// The example displays the following output:
//       Created __ComObject type from CLSID d055cba3-1f83-4bd7-ba19-
e22b1b8ec3c4
//       ExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {D055CBA3-1F83-4BD7-BA19-E22B1B8EC3C4} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
//
//       Created __ComObject type from CLSID 74f03346-a718-3516-ac78-
f351c7459ffb
//       NoExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {74F03346-A718-3516-AC78-F351C7459FFB} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
```

Instead, the GetTypeFromCLSID(Guid, String, Boolean) should only be used to retrieve the GUID of an unmanaged COM object, and the resulting Type object that is passed to the CreateInstance(Type) method must represent an unmanaged COM object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromCLSID(Guid, Boolean)

Gets the type associated with the specified class identifier (CLSID), specifying whether to throw an exception if an error occurs while loading the type.

```
C#
```

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromCLSID (Guid clsid, bool throwOnError);
```

Parameters

`clsid` Guid

The CLSID of the type to get.

`throwOnError` Boolean

`true` to throw any exception that occurs.

-or-

`false` to ignore any exception that occurs.

Returns

Type

`System.__ComObject` regardless of whether the CLSID is valid.

Attributes SupportedOSPlatformAttribute

Examples

The following example uses the CLSID of the Microsoft Word Application object to retrieve a COM type that represents the Microsoft Word application. It then instantiates the type by calling the Activator.CreateInstance method, and closes it by calling the Application.Quit method. An exception is thrown if an error occurs while loading the type.

C#

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

public class Example
{
   private const string WORD_CLSID = "{000209FF-0000-0000-C000-
000000000046}";

   public static void Main()
   {
```

```
        try {
           // Start an instance of the Word application.
           var word = Type.GetTypeFromCLSID(Guid.Parse(WORD_CLSID), true);
           Console.WriteLine("Instantiated Type object from CLSID {0}",
                             WORD_CLSID);
           Object wordObj = Activator.CreateInstance(word);
           Console.WriteLine("Instantiated {0}",
                             wordObj.GetType().FullName, WORD_CLSID);

           // Close Word.
           word.InvokeMember("Quit", BindingFlags.InvokeMethod, null,
                             wordObj, new object[] { 0, 0, false } );
        }
        catch (Exception) {
           Console.WriteLine("Unable to instantiate an object for {0}",
WORD_CLSID);
        }
    }
}
// The example displays the following output:
//    Instantiated Type object from CLSID {000209FF-0000-0000-C000-
000000000046}
//    Instantiated Microsoft.Office.Interop.Word.ApplicationClass
```

Remarks

The GetTypeFromCLSID method supports late-bound access to unmanaged COM objects from .NET Framework apps when you know the COM object's class identifier (CLSID). The class identifier for COM classes is defined in the HKEY_CLASSES_ROOT\CLSID key of the registry. You can retrieve the value of the IsCOMObject property to determine whether the type returned by this method is a COM object.

> **Tip**
>
> You can call the **GetTypeFromProgID** method for late-bound access to COM objects whose programmatic identifier (ProgID) you know.

Instantiating an unmanaged COM object from its CLSID is a two-step process:

1. Get a Type object that represents the `__ComObject` that corresponds to the CLSID by calling the GetTypeFromCLSID method.

2. Call the Activator.CreateInstance(Type) method to instantiate the COM object.

See the example for an illustration.

Exceptions such as OutOfMemoryException will be thrown when specifying `true` for `throwOnError`, but it will not fail for unregistered CLSIDs.

Notes to Callers

This method is intended for use when working with COM objects, not with .NET Framework objects. All managed objects, including those that are visible to COM (that is, their ComVisibleAttribute attribute is `true`) have a GUID that is returned by the GUID property. Although the method returns a Type object that corresponds to the GUID for .NET Framework objects, you can't use that Type object to create a type instance by calling the CreateInstance(Type) method, as the following example shows.

C#

```
using System;
using System.Runtime.InteropServices;

[assembly:ComVisible(true)]

// Define two classes, and assign one an explicit GUID.
[GuidAttribute("d055cba3-1f83-4bd7-ba19-e22b1b8ec3c4")]
public class ExplicitGuid
{ }

public class NoExplicitGuid
{ }

public class Example
{
   public static void Main()
   {
      Type explicitType = typeof(ExplicitGuid);
      Guid explicitGuid = explicitType.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type explicitCOM = Type.GetTypeFromCLSID(explicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        explicitCOM.Name, explicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        explicitType.Name, explicitCOM.Name,
                        explicitType.Equals(explicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(explicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
```

```
        }
        catch (COMException e) {
           Console.WriteLine("COM Exception:\n{0}\n", e.Message);
        }

        Type notExplicit = typeof(NoExplicitGuid);
        Guid notExplicitGuid = notExplicit.GUID;

        // Get type of ExplicitGuid from its GUID.
        Type notExplicitCOM = Type.GetTypeFromCLSID(notExplicitGuid);
        Console.WriteLine("Created {0} type from CLSID {1}",
                          notExplicitCOM.Name, notExplicitGuid);

        // Compare the two type objects.
        Console.WriteLine("{0} and {1} equal: {2}",
                          notExplicit.Name, notExplicitCOM.Name,
                          notExplicit.Equals(notExplicitCOM));

        // Instantiate an ExplicitGuid object.
        try {
           Object obj = Activator.CreateInstance(notExplicitCOM);
           Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
        }
        catch (COMException e) {
           Console.WriteLine("COM Exception:\n{0}\n", e.Message);
        }
    }
}
// The example displays the following output:
//       Created __ComObject type from CLSID d055cba3-1f83-4bd7-ba19-
e22b1b8ec3c4
//       ExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {D055CBA3-1F83-4BD7-BA19-E22B1B8EC3C4} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
//
//       Created __ComObject type from CLSID 74f03346-a718-3516-ac78-
f351c7459ffb
//       NoExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {74F03346-A718-3516-AC78-F351C7459FFB} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
```

Instead, the GetTypeFromCLSID(Guid, String, Boolean) should only be used to retrieve the GUID of an unmanaged COM object, and the resulting Type object that

is passed to the CreateInstance(Type) method must represent an unmanaged COM object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromCLSID(Guid, String)

Gets the type associated with the specified class identifier (CLSID) from the specified server.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromCLSID (Guid clsid, string? server);
```

Parameters

`clsid` Guid

The CLSID of the type to get.

`server` String

The server from which to load the type. If the server name is `null`, this method automatically reverts to the local machine.

Returns

Type

`System.__ComObject` regardless of whether the CLSID is valid.

Attributes SupportedOSPlatformAttribute

Examples

The following example uses the CLSID of the Microsoft Word Application object to retrieve a COM type that represents the Microsoft Word application from a server named computer17.central.contoso.com. It then instantiates the type by calling the Activator.CreateInstance method, and closes it by calling the Application.Quit method.

C#

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

public class Example
{
   private const string WORD_CLSID = "{000209FF-0000-0000-C000-
000000000046}";

   public static void Main()
   {
      // Start an instance of the Word application.
      var word = Type.GetTypeFromCLSID(Guid.Parse(WORD_CLSID),
"computer17.central.contoso.com");
      Console.WriteLine("Instantiated Type object from CLSID {0}",
                        WORD_CLSID);
      try {
         Object wordObj = Activator.CreateInstance(word);
         Console.WriteLine("Instantiated {0}",
                           wordObj.GetType().FullName, WORD_CLSID);

         // Close Word.
         word.InvokeMember("Quit", BindingFlags.InvokeMethod, null,
                           wordObj, new object[] { 0, 0, false } );
      }
      catch (COMException) {
         Console.WriteLine("Unable to instantiate object.");
      }
   }
}
// The example displays the following output:
//    Instantiated Type object from CLSID {000209FF-0000-0000-C000-
000000000046}
//    Instantiated Microsoft.Office.Interop.Word.ApplicationClass
```

Remarks

The GetTypeFromCLSID method supports late-bound access to unmanaged COM objects from .NET Framework apps when you know the COM object's class identifier

(CLSID). The class identifier for COM classes is defined in the HKEY_CLASSES_ROOT\CLSID key of the registry. You can retrieve the value of the IsCOMObject property to determine whether the type returned by this method is a COM object.

> **Tip**
>
> You can call the **GetTypeFromProgID** method for late-bound access to COM objects whose programmatic identifier (ProgID) you know.

Instantiating an unmanaged COM object from its CLSID is a two-step process:

1. Get a Type object that represents the `__ComObject` that corresponds to the CLSID by calling the GetTypeFromCLSID method.

2. Call the Activator.CreateInstance(Type) method to instantiate the COM object.

Notes to Callers

This method is intended for use when working with COM objects, not with .NET Framework objects. All managed objects, including those that are visible to COM (that is, their ComVisibleAttribute attribute is `true`) have a GUID that is returned by the GUID property. Although the method returns a Type object that corresponds to the GUID for .NET Framework objects, you can't use that Type object to create a type instance by calling the CreateInstance(Type) method, as the following example shows.

C#

```
using System;
using System.Runtime.InteropServices;

[assembly:ComVisible(true)]

// Define two classes, and assign one an explicit GUID.
[GuidAttribute("d055cba3-1f83-4bd7-ba19-e22b1b8ec3c4")]
public class ExplicitGuid
{ }

public class NoExplicitGuid
{ }

public class Example
{
   public static void Main()
   {
```

```
      Type explicitType = typeof(ExplicitGuid);
      Guid explicitGuid = explicitType.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type explicitCOM = Type.GetTypeFromCLSID(explicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        explicitCOM.Name, explicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        explicitType.Name, explicitCOM.Name,
                        explicitType.Equals(explicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(explicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
      }
      catch (COMException e) {
         Console.WriteLine("COM Exception:\n{0}\n", e.Message);
      }

      Type notExplicit = typeof(NoExplicitGuid);
      Guid notExplicitGuid = notExplicit.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type notExplicitCOM = Type.GetTypeFromCLSID(notExplicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        notExplicitCOM.Name, notExplicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        notExplicit.Name, notExplicitCOM.Name,
                        notExplicit.Equals(notExplicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(notExplicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
      }
      catch (COMException e) {
         Console.WriteLine("COM Exception:\n{0}\n", e.Message);
      }
   }
}
// The example displays the following output:
//       Created __ComObject type from CLSID d055cba3-1f83-4bd7-ba19-
e22b1b8ec3c4
//       ExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {D055CBA3-1F83-4BD7-BA19-E22B1B8EC3C4} failed due to the
following error:
```

```
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
//
//       Created __ComObject type from CLSID 74f03346-a718-3516-ac78-
f351c7459ffb
//       NoExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {74F03346-A718-3516-AC78-F351C7459FFB} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
```

Instead, the GetTypeFromCLSID(Guid, String, Boolean) should only be used to retrieve the GUID of an unmanaged COM object, and the resulting Type object that is passed to the CreateInstance(Type) method must represent an unmanaged COM object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromCLSID(Guid, String, Boolean)

Gets the type associated with the specified class identifier (CLSID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromCLSID (Guid clsid, string? server, bool
throwOnError);
```

Parameters

`clsid` Guid

The CLSID of the type to get.

`server` String

The server from which to load the type. If the server name is `null`, this method automatically reverts to the local machine.

`throwOnError` Boolean

`true` to throw any exception that occurs.

-or-

`false` to ignore any exception that occurs.

Returns

Type

`System.__ComObject` regardless of whether the CLSID is valid.

Attributes SupportedOSPlatformAttribute

Examples

The following example uses the CLSID of the Microsoft Word Application object to retrieve a COM type that represents the Microsoft Word application from a server named computer17.central.contoso.com. It then instantiates the type by calling the Activator.CreateInstance method, and closes it by calling the Application.Quit method. An exception is thrown if an error occurs while loading the type.

C#

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

public class Example
{
   private const string WORD_CLSID = "{000209FF-0000-0000-C000-
000000000046}";

   public static void Main()
   {
      try {
         // Start an instance of the Word application.
```

```
         var word = Type.GetTypeFromCLSID(Guid.Parse(WORD_CLSID),

"computer17.central.contoso.com",
                                          true);
         Console.WriteLine("Instantiated Type object from CLSID {0}",
                           WORD_CLSID);
         Object wordObj = Activator.CreateInstance(word);
         Console.WriteLine("Instantiated {0}",
                           wordObj.GetType().FullName, WORD_CLSID);

         // Close Word.
         word.InvokeMember("Quit", BindingFlags.InvokeMethod, null,
                           wordObj, new object[] { 0, 0, false } );
      }
      // The method can throw any of a variety of exceptions.
      catch (Exception e) {
         Console.WriteLine("{0}: Unable to instantiate an object for
{1}",
                           e.GetType().Name, WORD_CLSID);
      }
   }
}
// The example displays the following output:
//    Instantiated Type object from CLSID {000209FF-0000-0000-C000-
000000000046}
//    Instantiated Microsoft.Office.Interop.Word.ApplicationClass
```

Remarks

The GetTypeFromCLSID method supports late-bound access to unmanaged COM objects from .NET Framework apps when you know the COM object's class identifier (CLSID). The class identifier for COM classes is defined in the HKEY_CLASSES_ROOT\CLSID key of the registry. You can retrieve the value of the IsCOMObject property to determine whether the type returned by this method is a COM object.

> **Tip**
>
> You can call the **GetTypeFromProgID** method for late-bound access to COM objects whose programmatic identifier (ProgID) you know.

Instantiating an unmanaged COM object from its CLSID is a two-step process:

1. Get a Type object that represents the `__ComObject` that corresponds to the CLSID by calling the GetTypeFromCLSID method.

2. Call the Activator.CreateInstance(Type) method to instantiate the COM object.

Exceptions such as OutOfMemoryException will be thrown when specifying `true` for `throwOnError`, but it will not fail for unregistered CLSIDs.

Notes to Callers

This method is intended for use when working with COM objects, not with .NET Framework objects. All managed objects, including those that are visible to COM (that is, their ComVisibleAttribute attribute is `true`) have a GUID that is returned by the GUID property. Although the GetTypeFromCLSID(Guid, String, Boolean) method returns a Type object that corresponds to the GUID for a particular managed object, you can't use that Type object to create a type instance by calling the CreateInstance(Type) method, as the following example shows.

C#

```
using System;
using System.Runtime.InteropServices;

[assembly:ComVisible(true)]

// Define two classes, and assign one an explicit GUID.
[GuidAttribute("d055cba3-1f83-4bd7-ba19-e22b1b8ec3c4")]
public class ExplicitGuid
{ }

public class NoExplicitGuid
{ }

public class Example
{
   public static void Main()
   {
      Type explicitType = typeof(ExplicitGuid);
      Guid explicitGuid = explicitType.GUID;

      // Get type of ExplicitGuid from its GUID.
      Type explicitCOM = Type.GetTypeFromCLSID(explicitGuid);
      Console.WriteLine("Created {0} type from CLSID {1}",
                        explicitCOM.Name, explicitGuid);

      // Compare the two type objects.
      Console.WriteLine("{0} and {1} equal: {2}",
                        explicitType.Name, explicitCOM.Name,
                        explicitType.Equals(explicitCOM));

      // Instantiate an ExplicitGuid object.
      try {
         Object obj = Activator.CreateInstance(explicitCOM);
         Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
```

```
        }
        catch (COMException e) {
           Console.WriteLine("COM Exception:\n{0}\n", e.Message);
        }

        Type notExplicit = typeof(NoExplicitGuid);
        Guid notExplicitGuid = notExplicit.GUID;

        // Get type of ExplicitGuid from its GUID.
        Type notExplicitCOM = Type.GetTypeFromCLSID(notExplicitGuid);
        Console.WriteLine("Created {0} type from CLSID {1}",
                          notExplicitCOM.Name, notExplicitGuid);

        // Compare the two type objects.
        Console.WriteLine("{0} and {1} equal: {2}",
                          notExplicit.Name, notExplicitCOM.Name,
                          notExplicit.Equals(notExplicitCOM));

        // Instantiate an ExplicitGuid object.
        try {
           Object obj = Activator.CreateInstance(notExplicitCOM);
           Console.WriteLine("Instantiated a {0} object",
obj.GetType().Name);
        }
        catch (COMException e) {
           Console.WriteLine("COM Exception:\n{0}\n", e.Message);
        }
    }
}
// The example displays the following output:
//       Created __ComObject type from CLSID d055cba3-1f83-4bd7-ba19-
e22b1b8ec3c4
//       ExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {D055CBA3-1F83-4BD7-BA19-E22B1B8EC3C4} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
//
//       Created __ComObject type from CLSID 74f03346-a718-3516-ac78-
f351c7459ffb
//       NoExplicitGuid and __ComObject equal: False
//       COM Exception:
//       Retrieving the COM class factory for component with CLSID
//       {74F03346-A718-3516-AC78-F351C7459FFB} failed due to the
following error:
//       80040154 Class not registered
//       (Exception from HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
```

Instead, the GetTypeFromCLSID(Guid, String, Boolean) should only be used to retrieve the GUID of an unmanaged COM object, and the resulting Type object that

is passed to the CreateInstance(Type) method must represent an unmanaged COM object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeFromHandle(RuntimeTypeHandle) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type referenced by the specified type handle.

C#

```
public static Type? GetTypeFromHandle (RuntimeTypeHandle handle);
```

Parameters

`handle` RuntimeTypeHandle

The object that refers to the type.

Returns

Type

The type referenced by the specified RuntimeTypeHandle, or `null` if the Value property of `handle` is `null`.

Exceptions

TargetInvocationException

A class initializer is invoked and throws an exception.

Examples

The following example uses the GetTypeFromHandle method to get a Type object from a RuntimeTypeHandle provided by the GetTypeHandle method.

C#

```
MyClass1 myClass1 = new MyClass1();
 // Get the type referenced by the specified type handle.
Type myClass1Type = Type.GetTypeFromHandle(Type.GetTypeHandle(myClass1));
Console.WriteLine("The Names of the Attributes :"+myClass1Type.Attributes);
```

Remarks

The handles are valid only in the application domain in which they were obtained.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- RuntimeTypeHandle
- TypeHandle
- GetTypeHandle(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeFromProgID Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type associated with the specified program identifier (ProgID).

Overloads

Expand table

GetTypeFromProgID(String)	Gets the type associated with the specified program identifier (ProgID), returning null if an error is encountered while loading the Type.
GetTypeFromProgID(String, Boolean)	Gets the type associated with the specified program identifier (ProgID), specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromProgID(String, String)	Gets the type associated with the specified program identifier (progID) from the specified server, returning null if an error is encountered while loading the type.
GetTypeFromProgID(String, String, Boolean)	Gets the type associated with the specified program identifier (progID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.

GetTypeFromProgID(String)

Gets the type associated with the specified program identifier (ProgID), returning null if an error is encountered while loading the Type.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromProgID (string progID);
```

Parameters

`progID` String

The ProgID of the type to get.

Returns

Type

The type associated with the specified ProgID, if `progID` is a valid entry in the registry and a type is associated with it; otherwise, `null`.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentException

`progID` is `null`.

Remarks

This method is provided for COM support. ProgIDs are not used in the Microsoft .NET Framework because they have been superseded by the concept of namespace.

See also

- GetProgIdForType(Type)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromProgID(String, Boolean)

Gets the type associated with the specified program identifier (ProgID), specifying whether to throw an exception if an error occurs while loading the type.

```
C#

[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromProgID (string progID, bool throwOnError);
```

Parameters

progID String

The ProgID of the type to get.

throwOnError Boolean

`true` to throw any exception that occurs.

-or-

`false` to ignore any exception that occurs.

Returns

Type

The type associated with the specified program identifier (ProgID), if `progID` is a valid entry in the registry and a type is associated with it; otherwise, `null`.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentException

`progID` is `null`.

COMException

The specified ProgID is not registered.

Examples

The following example retrieves a type by passing a ProgID, specifying whether to throw an exception if the ProgID is invalid. The example then displays the ClassID related to the ProgID, along with any applicable exception message.

C#

```
using System;
class MainApp
{
    public static void Main()
    {
        try
        {
            // Use the ProgID HKEY_CLASSES_ROOT\DirControl.DirList.1.
            string myString1 ="DIRECT.ddPalette.3";
            // Use a nonexistent ProgID WrongProgID.
            string myString2 ="WrongProgID";
            // Make a call to the method to get the type information of
the given ProgID.
            Type myType1 =Type.GetTypeFromProgID(myString1,true);
            Console.WriteLine("GUID for ProgID DirControl.DirList.1 is
{0}.", myType1.GUID);
            // Throw an exception because the ProgID is invalid and the
throwOnError
            // parameter is set to True.
            Type myType2 =Type.GetTypeFromProgID(myString2,true);
        }
        catch(Exception e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Source: {0}", e.Source);
            Console.WriteLine("Message: {0}", e.Message);
        }
    }
}
```

Remarks

This method is provided for COM support. Program IDs are not used in Microsoft .NET Framework because they have been superseded by the concept of namespace.

See also

- GetProgIdForType(Type)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromProgID(String, String)

Gets the type associated with the specified program identifier (progID) from the specified server, returning null if an error is encountered while loading the type.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromProgID (string progID, string? server);
```

Parameters

progID String

The progID of the type to get.

server String

The server from which to load the type. If the server name is `null`, this method automatically reverts to the local machine.

Returns

Type

The type associated with the specified program identifier (progID), if `progID` is a valid entry in the registry and a type is associated with it; otherwise, `null`.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentException

`prodID` is `null`.

Examples

The following example retrieves a type by passing a ProgID and server name. The example then displays the ClassID related to the ProgID, or throws an exception if the ProgID or the server name is invalid.

C#

```
using System;
class MainApp
{
    public static void Main()
    {
        try
        {
            // Use the ProgID
localhost\HKEY_CLASSES_ROOT\DirControl.DirList.1.
            string theProgramID ="DirControl.DirList.1";
            // Use the server name localhost.
            string theServer="localhost";
            // Make a call to the method to get the type information for
the given ProgID.
            Type myType =Type.GetTypeFromProgID(theProgramID,theServer);
            if(myType==null)
            {
                throw new Exception("Invalid ProgID or Server.");
            }
            Console.WriteLine("GUID for ProgID DirControl.DirList.1 is
{0}.", myType.GUID);
        }
        catch(Exception e)
        {
            Console.WriteLine("An exception occurred.");
            Console.WriteLine("Source: {0}" , e.Source);
            Console.WriteLine("Message: {0}" , e.Message);
        }
    }
}
```

Remarks

This method is provided for COM support. Program IDs are not used in Microsoft .NET Framework because they have been superseded by the concept of namespace.

See also

- GetProgIdForType(Type)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

GetTypeFromProgID(String, String, Boolean)

Gets the type associated with the specified program identifier (progID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.

C#

```
[System.Runtime.Versioning.SupportedOSPlatform("windows")]
public static Type? GetTypeFromProgID (string progID, string? server,
bool throwOnError);
```

Parameters

`progID` String

The progID of the Type to get.

`server` String

The server from which to load the type. If the server name is `null`, this method automatically reverts to the local machine.

`throwOnError` Boolean

`true` to throw any exception that occurs.

-or-

`false` to ignore any exception that occurs.

Returns

Type

The type associated with the specified program identifier (progID), if `progID` is a valid entry in the registry and a type is associated with it; otherwise, `null`.

Attributes SupportedOSPlatformAttribute

Exceptions

ArgumentException

`progID` is `null`.

COMException

The specified progID is not registered.

Examples

The following example retrieves a type by passing a ProgID and server name. The example then displays the ClassID related to the ProgID, specifying whether to throw an exception if the ProgID or the server name is invalid.

C#

```
using System;
class MainApp
{
    public static void Main()
    {
        try
        {
            // Use server localhost.
            string theServer="localhost";
            // Use  ProgID HKEY_CLASSES_ROOT\DirControl.DirList.1.
            string myString1 ="DirControl.DirList.1";
            // Use a wrong ProgID WrongProgID.
            string myString2 ="WrongProgID";
            // Make a call to the method to get the type information for
the given ProgID.
            Type myType1
=Type.GetTypeFromProgID(myString1,theServer,true);
            Console.WriteLine("GUID for ProgID DirControl.DirList.1 is
{0}.", myType1.GUID);
            // Throw an exception because the ProgID is invalid and the
throwOnError
            // parameter is set to True.
            Type myType2 =Type.GetTypeFromProgID(myString2, theServer,
true);
        }
        catch(Exception e)
```

```
        {
            Console.WriteLine("An exception occurred. The ProgID is
wrong.");
            Console.WriteLine("Source: {0}" , e.Source);
            Console.WriteLine("Message: {0}" , e.Message);
        }
    }
}
```

Remarks

This method is provided for COM support. Program IDs are not used in Microsoft .NET Framework because they have been superseded by the concept of namespace.

See also

- GetProgIdForType(Type)
- TypeLoadException

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.GetTypeHandle(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the handle for the Type of a specified object.

C#

```
public static RuntimeTypeHandle GetTypeHandle (object o);
```

Parameters

o Object

The object for which to get the type handle.

Returns

RuntimeTypeHandle

The handle for the Type of the specified Object.

Exceptions

ArgumentNullException

`o` is `null`.

Examples

The following example defines the class `MyClass1`, gets an instance of it, and retrieves the runtime handle of the object.

C#

```
using System;
using System.Reflection;

public class MyClass1
```

```
{
    private int x=0;
    public int MyMethod()
    {
        return x;
    }
}

public class MyClass2
{
    public static void Main()
    {
        MyClass1 myClass1 = new MyClass1();

        // Get the RuntimeTypeHandle from an object.
        RuntimeTypeHandle myRTHFromObject = Type.GetTypeHandle(myClass1);
        // Get the RuntimeTypeHandle from a type.
        RuntimeTypeHandle myRTHFromType = typeof(MyClass1).TypeHandle;

        Console.WriteLine("\nmyRTHFromObject.Value:  {0}",
myRTHFromObject.Value);
        Console.WriteLine("myRTHFromObject.GetType():  {0}",
myRTHFromObject.GetType());
        Console.WriteLine("Get the type back from the handle...");
        Console.WriteLine("Type.GetTypeFromHandle(myRTHFromObject):  {0}",
            Type.GetTypeFromHandle(myRTHFromObject));

        Console.WriteLine("\nmyRTHFromObject.Equals(myRTHFromType):  {0}",
            myRTHFromObject.Equals(myRTHFromType));

        Console.WriteLine("\nmyRTHFromType.Value:  {0}",
myRTHFromType.Value);
        Console.WriteLine("myRTHFromType.GetType():  {0}",
myRTHFromType.GetType());
        Console.WriteLine("Get the type back from the handle...");
        Console.WriteLine("Type.GetTypeFromHandle(myRTHFromType):  {0}",
            Type.GetTypeFromHandle(myRTHFromType));
    }
}

/* This code example produces output similar to the following:

myRTHFromObject.Value:  799464
myRTHFromObject.GetType():  System.RuntimeTypeHandle
Get the type back from the handle...
Type.GetTypeFromHandle(myRTHFromObject):  MyClass1

myRTHFromObject.Equals(myRTHFromType):  True

myRTHFromType.Value:  799464
myRTHFromType.GetType():  System.RuntimeTypeHandle
Get the type back from the handle...
Type.GetTypeFromHandle(myRTHFromType):  MyClass1
 */
```

Remarks

The handles are valid only in the application domain in which they were obtained.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RuntimeTypeHandle
- TypeHandle
- GetTypeFromHandle(RuntimeTypeHandle)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.HasElementTypeImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the HasElementType property and determines whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.

C#

```
protected abstract bool HasElementTypeImpl ();
```

Returns

Boolean

`true` if the Type is an array, a pointer, or is passed by reference; otherwise, `false`.

Examples

The following example defines the class `MyTypeDelegator`, which overrides the `HasElementTypeImpl` method. The main class checks for the `HasElementType` property and displays the element type.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegator : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;
    public MyTypeDelegator(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override Type.HasElementTypeImpl().
    protected override bool HasElementTypeImpl()
    {
```

```
        // Determine whether the type is an array.
        if(myType.IsArray)
        {
            myElementType = "array";
            return true;
        }
        // Determine whether the type is a reference.
        if(myType.IsByRef)
        {
            myElementType = "reference";
            return true;
        }
        // Determine whether the type is a pointer.
        if(myType.IsPointer)
        {
            myElementType = "pointer";
            return true;
        }
        // Return false if the type is not a reference, array, or pointer
type.
        return false;
    }
}
public class Type_HasElementTypeImpl
{
    public static void Main()
    {
        try
        {
            int myInt = 0 ;
            int[] myArray = new int[5];
            MyTypeDelegator myType = new MyTypeDelegator(myArray.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            Console.WriteLine("\nDetermine whether a variable is an array,
pointer, or reference type.\n");
            if( myType.HasElementType)
                Console.WriteLine("The type of myArray is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myArray is not an array, pointer, or
reference type.");
            myType = new MyTypeDelegator(myInt.GetType());
            // Determine whether myType is an array, pointer, reference
type.
            if( myType.HasElementType)
                Console.WriteLine("The type of myInt is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myInt is not an array, pointer, or
reference type.");
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
```

```
        }
    }
}
```

Remarks

For example, Type.GetType("Int32[]").HasElementTypeImpl returns `true`, but Type.GetType("Int32").HasElementTypeImpl returns `false`. HasElementTypeImpl also returns `true` for "Int32*" and "Int32&".

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- HasElementType
- IsArray
- IsPointer
- IsByRef
- GetElementType()
- GetType()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.InvokeMember Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Invokes a specific member of the current Type.

Overloads

Expand table

InvokeMember(String, BindingFlags, Binder, Object, Object[])	Invokes the specified member, using the specified binding constraints and matching the specified argument list.
InvokeMember(String, BindingFlags, Binder, Object, Object[], CultureInfo)	Invokes the specified member, using the specified binding constraints and matching the specified argument list and culture.
InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])	When overridden in a derived class, invokes the specified member, using the specified binding constraints and matching the specified argument list, modifiers and culture.

InvokeMember(String, BindingFlags, Binder, Object, Object[])

Invokes the specified member, using the specified binding constraints and matching the specified argument list.

C#

```
public object? InvokeMember (string name, System.Reflection.BindingFlags
invokeAttr, System.Reflection.Binder? binder, object? target, object?[]?
args);
```

Parameters

`name` String

The string containing the name of the constructor, method, property, or field member to invoke.

-or-

An empty string ("") to invoke the default member.

-or-

For `IDispatch` members, a string representing the DispID, for example "[DispID=3]".

`invokeAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted. The access can be one of the `BindingFlags` such as `Public`, `NonPublic`, `Private`, `InvokeMethod`, `GetField`, and so on. The type of lookup need not be specified. If the type of lookup is omitted, `BindingFlags.Public` | `BindingFlags.Instance` | `BindingFlags.Static` are used.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder. Note that explicitly defining a Binder object may be required for successfully invoking method overloads with variable arguments.

`target` Object

The object on which to invoke the specified member.

`args` Object[]

An array containing the arguments to pass to the member to invoke.

Returns

Object

An object representing the return value of the invoked member.

Exceptions

ArgumentNullException

`invokeAttr` does not contain `CreateInstance` and `name` is `null`.

ArgumentException

`invokeAttr` is not a valid BindingFlags attribute.

-or-

`invokeAttr` does not contain one of the following binding flags: `InvokeMethod`, `CreateInstance`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains `CreateInstance` combined with `InvokeMethod`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains both `GetField` and `SetField`.

-or-

`invokeAttr` contains both `GetProperty` and `SetProperty`.

-or-

`invokeAttr` contains `InvokeMethod` combined with `SetField` or `SetProperty`.

-or-

`invokeAttr` contains `SetField` and `args` has more than one element.

-or-

This method is called on a COM object and one of the following binding flags was not passed in: `BindingFlags.InvokeMethod`, `BindingFlags.GetProperty`, `BindingFlags.SetProperty`, `BindingFlags.PutDispProperty`, or `BindingFlags.PutRefDispProperty`.

-or-

One of the named parameter arrays contains a string that is `null`.

MethodAccessException

The specified member is a class initializer.

MissingFieldException

The field or property cannot be found.

MissingMethodException

No method can be found that matches the arguments in `args`.

-or-

The current Type object represents a type that contains open type parameters, that is, ContainsGenericParameters returns `true`.

TargetException

The specified member cannot be invoked on `target`.

AmbiguousMatchException

More than one method matches the binding criteria.

NotSupportedException

The .NET Compact Framework does not currently support this method.

InvalidOperationException

The method represented by `name` has one or more unspecified generic type parameters. That is, the method's ContainsGenericParameters property returns `true`.

Examples

The following example uses `InvokeMember` to access members of a type.

C#

```
using System;
using System.Reflection;

// This sample class has a field, constructor, method, and property.
class MyType
{
    Int32 myField;
    public MyType(ref Int32 x) {x *= 5;}
    public override String ToString() {return myField.ToString();}
    public Int32 MyProp
    {
        get {return myField;}
```

```
        set
        {
            if (value < 1)
                throw new ArgumentOutOfRangeException("value", value,
"value must be > 0");
            myField = value;
        }
    }
}

class MyApp
{
    static void Main()
    {
        Type t = typeof(MyType);
        // Create an instance of a type.
        Object[] args = new Object[] {8};
        Console.WriteLine("The value of x before the constructor is
called is {0}.", args[0]);
        Object obj = t.InvokeMember(null,
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.CreateInstance, null,
null, args);
        Console.WriteLine("Type: " + obj.GetType().ToString());
        Console.WriteLine("The value of x after the constructor returns
is {0}.", args[0]);

        // Read and write to a field.
        t.InvokeMember("myField",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.SetField, null, obj, new
Object[] {5});
        Int32 v = (Int32) t.InvokeMember("myField",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.GetField, null, obj,
null);
        Console.WriteLine("myField: " + v);

        // Call a method.
        String s = (String) t.InvokeMember("ToString",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.InvokeMethod, null, obj,
null);
        Console.WriteLine("ToString: " + s);

        // Read and write a property. First, attempt to assign an
        // invalid value; then assign a valid value; finally, get
        // the value.
        try
        {
            // Assign the value zero to MyProp. The Property Set
```

```
            // throws an exception, because zero is an invalid value.
            // InvokeMember catches the exception, and throws
            // TargetInvocationException. To discover the real cause
            // you must catch TargetInvocationException and examine
            // the inner exception.
            t.InvokeMember("MyProp",
                BindingFlags.DeclaredOnly |
                BindingFlags.Public | BindingFlags.NonPublic |
                BindingFlags.Instance | BindingFlags.SetProperty, null,
obj, new Object[] {0});
        }
        catch (TargetInvocationException e)
        {
            // If the property assignment failed for some unexpected
            // reason, rethrow the TargetInvocationException.
            if (e.InnerException.GetType() !=
                typeof(ArgumentOutOfRangeException))
                throw;
            Console.WriteLine("An invalid value was assigned to
MyProp.");
        }
        t.InvokeMember("MyProp",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.SetProperty, null, obj,
new Object[] {2});
        v = (Int32) t.InvokeMember("MyProp",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.GetProperty, null, obj,
null);
        Console.WriteLine("MyProp: " + v);
    }
}
```

Remarks

> **Note**
>
> You cannot use **InvokeMember** to invoke a generic method.

The following BindingFlags filter flags can be used to define which members to include in the search:

- Specify `BindingFlags.Public` to include public members in the search.
- Specify `BindingFlags.NonPublic` to include non-public members (that is, private and protected members) in the search.

- Specify `BindingFlags.FlattenHierarchy` to include static members up the hierarchy.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

The following BindingFlags invocation flags can be used to denote what action to take with the member:

- `CreateInstance` to invoke a constructor. `name` is ignored. Not valid with other invocation flags.
- `InvokeMethod` to invoke a method, but not a constructor or a type initializer. Not valid with `SetField` or `SetProperty`. If `InvokeMethod` is specified by itself, `BindingFlags.Public`, `BindingFlags.Instance`, and `BindingFlags.Static` are automatically included.
- `GetField` to get the value of a field. Not valid with `SetField`.
- `SetField` to set the value of a field. Not valid with `GetField`.
- `GetProperty` to get a property. Not valid with `SetProperty`.
- `SetProperty` to set a property. Not valid with `GetProperty`.

See System.Reflection.BindingFlags for more information.

A method will be invoked if both of the following conditions are true:

- The number of parameters in the method declaration equals the number of arguments in the `args` array (unless default arguments are defined on the member and `BindingFlags.OptionalParamBinding` is specified).
- The type of each argument can be converted by the binder to the type of the parameter.

The binder will find all of the matching methods. These methods are found based upon the type of binding requested (BindingFlags values `InvokeMethod`, `GetProperty`, and so on). The set of methods is filtered by the name, number of arguments, and a set of search modifiers defined in the binder.

After the method is selected, it is invoked. Accessibility is checked at that point. The search may control which set of methods are searched based upon the accessibility attribute associated with the method. The Binder.BindToMethod method of the Binder class is responsible for selecting the method to be invoked. The default binder selects the most specific match.

Access restrictions are ignored for fully trusted code; that is, private constructors, methods, fields, and properties can be accessed and invoked through System.Reflection whenever the code is fully trusted.

You can use `Type.InvokeMember` to set a field to a particular value by specifying BindingFlags.SetField. For example, if you want to set a public instance field named F on class C, and F is a `String`, you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[]
{"strings new value"});
```

If F is a `String[]`, you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {new
String[]{"a","z","c","d"}});
```

which will initialize the field F to this new array. You can also use `Type.InvokeMember` to set a position in an array by supplying the index of the value and then the next value by using code such as the following:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {1,
"b"});
```

This will change string "z" in the array that F holds to string "b".

When you invoke an `IDispatch` member, you can specify the DispID instead of the member name, using the string format "[DispID=##]". For example, if the DispID of MyComMethod is 3, you can specify the string "[DispID=3]" instead of "MyComMethod". Invoking a member by DispID is faster than looking up the member by name. In complex aggregation scenarios, the DispID is sometimes the only way to invoke the desired member.

> **Note**
>
> Starting with the .NET Framework 2.0 Service Pack 1, this method can be used to access non-public members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the

grant set of the non-public members is restricted to the caller's grant set, or a subset thereof. (See **Security Considerations for Reflection**.)

To use this functionality, your application should target the .NET Framework 3.5 or later.

See also

- String
- Binder
- DefaultBinder
- BindingFlags
- ParameterModifier
- ParameterAttributes
- CultureInfo
- ReflectionPermission

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InvokeMember(String, BindingFlags, Binder, Object, Object[], CultureInfo)

Invokes the specified member, using the specified binding constraints and matching the specified argument list and culture.

C#

```
public object? InvokeMember (string name, System.Reflection.BindingFlags
invokeAttr, System.Reflection.Binder? binder, object? target, object?[]?
args, System.Globalization.CultureInfo? culture);
```

Parameters

`name` String

The string containing the name of the constructor, method, property, or field member to invoke.

-or-

An empty string ("") to invoke the default member.

-or-

For `IDispatch` members, a string representing the DispID, for example "[DispID=3]".

`invokeAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted. The access can be one of the `BindingFlags` such as `Public`, `NonPublic`, `Private`, `InvokeMethod`, `GetField`, and so on. The type of lookup need not be specified. If the type of lookup is omitted, `BindingFlags.Public` | `BindingFlags.Instance` | `BindingFlags.Static` are used.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (`Nothing` in Visual Basic), to use the DefaultBinder. Note that explicitly defining a Binder object may be required for successfully invoking method overloads with variable arguments.

`target` Object

The object on which to invoke the specified member.

`args` Object[]

An array containing the arguments to pass to the member to invoke.

`culture` CultureInfo

The object representing the globalization locale to use, which may be necessary for locale-specific conversions, such as converting a numeric String to a Double.

-or-

A null reference (`Nothing` in Visual Basic) to use the current thread's CultureInfo.

Returns

Object

An object representing the return value of the invoked member.

Exceptions

ArgumentNullException

`invokeAttr` does not contain `CreateInstance` and `name` is `null`.

ArgumentException

`invokeAttr` is not a valid BindingFlags attribute.

-or-

`invokeAttr` does not contain one of the following binding flags: `InvokeMethod`, `CreateInstance`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains `CreateInstance` combined with `InvokeMethod`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains both `GetField` and `SetField`.

-or-

`invokeAttr` contains both `GetProperty` and `SetProperty`.

-or-

`invokeAttr` contains `InvokeMethod` combined with `SetField` or `SetProperty`.

-or-

`invokeAttr` contains `SetField` and `args` has more than one element.

-or-

This method is called on a COM object and one of the following binding flags was not passed in: `BindingFlags.InvokeMethod`, `BindingFlags.GetProperty`,

`BindingFlags.SetProperty`, `BindingFlags.PutDispProperty`, or `BindingFlags.PutRefDispProperty`.

-or-

One of the named parameter arrays contains a string that is `null`.

MethodAccessException
The specified member is a class initializer.

MissingFieldException
The field or property cannot be found.

MissingMethodException
No method can be found that matches the arguments in `args`.

-or-

The current Type object represents a type that contains open type parameters, that is, ContainsGenericParameters returns `true`.

TargetException
The specified member cannot be invoked on `target`.

AmbiguousMatchException
More than one method matches the binding criteria.

InvalidOperationException
The method represented by `name` has one or more unspecified generic type parameters. That is, the method's ContainsGenericParameters property returns `true`.

Remarks

Although the default binder does not process CultureInfo (the `culture` parameter), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `culture`.

> ⓘ **Note**
>
> You cannot use **InvokeMember** to invoke a generic method.

The following BindingFlags filter flags can be used to define which members to include in the search:

- Specify `BindingFlags.Public` to include public members in the search.
- Specify `BindingFlags.NonPublic` to include non-public members (that is, private, internal, and protected members) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include static members up the hierarchy.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.
- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

The following BindingFlags invocation flags can be used to denote what action to take with the member:

- `CreateInstance` to invoke a constructor. `name` is ignored. Not valid with other invocation flags.
- `InvokeMethod` to invoke a method, but not a constructor or a type initializer. Not valid with `SetField` or `SetProperty`. If `InvokeMethod` is specified by itself, `BindingFlags.Public`, `BindingFlags.Instance`, and `BindingFlags.Static` are automatically included.
- `GetField` to get the value of a field. Not valid with `SetField`.
- `SetField` to set the value of a field. Not valid with `GetField`.
- `GetProperty` to get a property. Not valid with `SetProperty`.
- `SetProperty` to set a property. Not valid with `GetProperty`.

See System.Reflection.BindingFlags for more information.

A method will be invoked if both of the following conditions are true:

- The number of parameters in the method declaration equals the number of arguments in the `args` array (unless default arguments are defined on the member and `BindingFlags.OptionalParamBinding` is specified).

- The type of each argument can be converted by the binder to the type of the parameter.

The binder will find all of the matching methods. These methods are found based upon the type of binding requested (BindingFlags values `InvokeMethod`, `GetProperty`, and so on). The set of methods is filtered by the name, number of arguments, and a set of search modifiers defined in the binder.

After the method is selected, it is invoked. Accessibility is checked at that point. The search may control which set of methods are searched based upon the accessibility attribute associated with the method. The Binder.BindToMethod method of the Binder class is responsible for selecting the method to be invoked. The default binder selects the most specific match.

Access restrictions are ignored for fully trusted code; that is, private constructors, methods, fields, and properties can be accessed and invoked through Reflection whenever the code is fully trusted.

You can use `Type.InvokeMember` to set a field to a particular value by specifying BindingFlags.SetField. For example, if you want to set a public instance field named F on class C, and F is a `String` you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[]
{"strings new value"}, null);
```

If F is a `String[]`, you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {new
String[]{"a","z","c","d"}}, null);
```

which will initialize the field F to this new array. You can also use `Type.InvokeMember` to set a position in an array by supplying the index of the value and then the next value by using code such as the following:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {1,
"b"}, null);
```

This will change string "z" in the array that F holds to string "b".

When you invoke an `IDispatch` member you can specify the DispID instead of the member name, using the string format "[DispID=##]". For example, if the DispID of MyComMethod is 3, you can specify the string "[DispID=3]" instead of "MyComMethod". Invoking a member by DispID is faster than looking up the

member by name. In complex aggregation scenarios, the DispID is sometimes the only way to invoke the desired member.

> **Note**
>
> Starting with the .NET Framework 2.0 Service Pack 1, this method can be used to access non-public members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the non-public members is restricted to the caller's grant set, or a subset thereof. (See **Security Considerations for Reflection**.)
>
> To use this functionality, your application should target the .NET Framework 3.5 or later.

See also

- String
- Binder
- DefaultBinder
- BindingFlags
- ParameterModifier
- ParameterAttributes
- CultureInfo
- ReflectionPermission

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[],

CultureInfo, String[])

When overridden in a derived class, invokes the specified member, using the specified binding constraints and matching the specified argument list, modifiers and culture.

```
C#

public abstract object? InvokeMember (string name,
System.Reflection.BindingFlags invokeAttr, System.Reflection.Binder?
binder, object? target, object?[]? args,
System.Reflection.ParameterModifier[]? modifiers,
System.Globalization.CultureInfo? culture, string[]? namedParameters);
```

Parameters

`name` String

The string containing the name of the constructor, method, property, or field member to invoke.

-or-

An empty string ("") to invoke the default member.

-or-

For `IDispatch` members, a string representing the DispID, for example "[DispID=3]".

`invokeAttr` BindingFlags

A bitwise combination of the enumeration values that specify how the search is conducted. The access can be one of the `BindingFlags` such as `Public`, `NonPublic`, `Private`, `InvokeMethod`, `GetField`, and so on. The type of lookup need not be specified. If the type of lookup is omitted, `BindingFlags.Public` | `BindingFlags.Instance` | `BindingFlags.Static` are used.

`binder` Binder

An object that defines a set of properties and enables binding, which can involve selection of an overloaded method, coercion of argument types, and invocation of a member through reflection.

-or-

A null reference (Nothing in Visual Basic), to use the DefaultBinder. Note that explicitly defining a Binder object may be required for successfully invoking method overloads with variable arguments.

`target` Object

The object on which to invoke the specified member.

`args` Object[]

An array containing the arguments to pass to the member to invoke.

`modifiers` ParameterModifier[]

An array of ParameterModifier objects representing the attributes associated with the corresponding element in the `args` array. A parameter's associated attributes are stored in the member's signature.

The default binder processes this parameter only when calling a COM component.

`culture` CultureInfo

The CultureInfo object representing the globalization locale to use, which may be necessary for locale-specific conversions, such as converting a numeric String to a Double.

-or-

A null reference (`Nothing` in Visual Basic) to use the current thread's CultureInfo.

`namedParameters` String[]

An array containing the names of the parameters to which the values in the `args` array are passed.

Returns

Object

An object representing the return value of the invoked member.

Implements

InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])

Exceptions

ArgumentNullException

`invokeAttr` does not contain `CreateInstance` and `name` is `null`.

ArgumentException

`args` and `modifiers` do not have the same length.

-or-

`invokeAttr` is not a valid BindingFlags attribute.

-or-

`invokeAttr` does not contain one of the following binding flags: `InvokeMethod`, `CreateInstance`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains `CreateInstance` combined with `InvokeMethod`, `GetField`, `SetField`, `GetProperty`, or `SetProperty`.

-or-

`invokeAttr` contains both `GetField` and `SetField`.

-or-

`invokeAttr` contains both `GetProperty` and `SetProperty`.

-or-

`invokeAttr` contains `InvokeMethod` combined with `SetField` or `SetProperty`.

-or-

`invokeAttr` contains `SetField` and `args` has more than one element.

-or-

The named parameter array is larger than the argument array.

-or-

This method is called on a COM object and one of the following binding flags was not passed in: `BindingFlags.InvokeMethod`, `BindingFlags.GetProperty`,

`BindingFlags.SetProperty`, `BindingFlags.PutDispProperty`, or `BindingFlags.PutRefDispProperty`.

-or-

One of the named parameter arrays contains a string that is `null`.

MethodAccessException
The specified member is a class initializer.

MissingFieldException
The field or property cannot be found.

MissingMethodException
No method can be found that matches the arguments in `args`.

-or-

No member can be found that has the argument names supplied in `namedParameters`.

-or-

The current Type object represents a type that contains open type parameters, that is, ContainsGenericParameters returns `true`.

TargetException
The specified member cannot be invoked on `target`.

AmbiguousMatchException
More than one method matches the binding criteria.

InvalidOperationException
The method represented by `name` has one or more unspecified generic type parameters. That is, the method's ContainsGenericParameters property returns `true`.

Remarks

`InvokeMember` calls a constructor member or a method member, gets or sets a property member, gets or sets a data field member, or gets or sets an element of an array member.

> **Note**
>
> You cannot use **InvokeMember** to invoke a generic method.

When you invoke an `IDispatch` member you can specify the DispID instead of the member name, using the string format "[DispID=##]". For example, if the DispID of MyComMethod is 3, you can specify the string "[DispID=3]" instead of "MyComMethod". Invoking a member by DispID is faster than looking up the member by name. In complex aggregation scenarios, the DispID is sometimes the only way to invoke the desired member.

Although the default binder does not process ParameterModifier or CultureInfo (the `modifiers` and `culture` parameters), you can use the abstract System.Reflection.Binder class to write a custom binder that does process `modifiers` and `culture`. `ParameterModifier` is only used when calling through COM interop, and only parameters that are passed by reference are handled.

Each parameter in the `namedParameters` array gets the value in the corresponding element in the `args` array. If the length of `args` is greater than the length of `namedParameters`, the remaining argument values are passed in order.

The `namedParameters` array can be used to change the order of arguments in an input array. For example, given the method `M(string a, int b)` (`M(ByVal a As String, ByVal b As Integer)` in Visual Basic) and the input array `{ 42, "x" }`, the input array can be passed unchanged to `args` if the array `{ "b", "a" }` is supplied for `namedParameters`.

The following BindingFlags filter flags can be used to define which members to include in the search:

- Specify `BindingFlags.Public` to include public members in the search.
- Specify `BindingFlags.NonPublic` to include non-public members (that is, private, internal, and protected members) in the search.
- Specify `BindingFlags.FlattenHierarchy` to include static members up the hierarchy.

The following BindingFlags modifier flags can be used to change how the search works:

- `BindingFlags.IgnoreCase` to ignore the case of `name`.

- `BindingFlags.DeclaredOnly` to search only the members declared on the Type, not members that were simply inherited.

The following BindingFlags invocation flags can be used to denote what action to take with the member:

- `CreateInstance` to invoke a constructor. `name` is ignored. Not valid with other invocation flags.
- `InvokeMethod` to invoke a method, but not a constructor or a type initializer. Not valid with `SetField` or `SetProperty`. If `InvokeMethod` is specified by itself, `BindingFlags.Public`, `BindingFlags.Instance`, and `BindingFlags.Static` are automatically included.
- `GetField` to get the value of a field. Not valid with `SetField`.
- `SetField` to set the value of a field. Not valid with `GetField`.
- `GetProperty` to get a property. Not valid with `SetProperty`.
- `SetProperty` to set a property. Not valid with `GetProperty`.

See System.Reflection.BindingFlags for more information.

A method will be invoked if both of the following conditions are true:

- The number of parameters in the method declaration equals the number of arguments in the `args` array (unless default arguments are defined on the member and `BindingFlags.OptionalParamBinding` is specified).
- The type of each argument can be converted by the binder to the type of the parameter.

The binder will find all of the matching methods. These methods are found based upon the type of binding requested (BindingFlags values `InvokeMethod`, `GetProperty`, and so on). The set of methods is filtered by the name, number of arguments, and a set of search modifiers defined in the binder.

After the method is selected, it is invoked. Accessibility is checked at that point. The search may control which set of methods are searched based upon the accessibility attribute associated with the method. The Binder.BindToMethod method of the Binder class is responsible for selecting the method to be invoked. The default binder selects the most specific match.

`InvokeMember` can be used to invoke methods with parameters that have default values. To bind to these methods, Reflection requires BindingFlags.OptionalParamBinding to be specified. For a parameter that has a default value, you can either supply a different value, or supply Missing.Value to use the default value.

For example, consider a method such as MyMethod(int x, float y = 2.0). To invoke this method with only the first argument as MyMethod(4), pass one of the above binding flags and pass two arguments, namely, 4 for the first argument and `Missing.Value` for the second argument. Unless you use `Missing.Value`, you may not omit optional parameters with the `Invoke` method. If you must do so, use `InvokeMember` instead.

Access restrictions are ignored for fully trusted code; that is, private constructors, methods, fields, and properties can be accessed and invoked through System.Reflection whenever the code is fully trusted.

You can use `Type.InvokeMember` to set a field to a particular value by specifying BindingFlags.SetField. For example, if you want to set a public instance field named F on class C, and F is a `String`, you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[]
{"strings new value"}, null, null, null);
```

If F is a `String[]`, you can use code such as:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {new
String[]{"a","z","c","d"}}, null, null, null);
```

which will initialize the field F to this new array. You can also use `Type.InvokeMember` to set a position in an array by supplying the index of the value and then the next value by using code such as the following:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, c, new Object[] {1,
"b"}, null, null, null);
```

This will change string "z" in the array that F holds to string "b".

> **Note**
>
> Starting with the .NET Framework 2.0 Service Pack 1, this method can be used to access non-public members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the

grant set of the non-public members is restricted to the caller's grant set, or a subset thereof. (See **Security Considerations for Reflection**.)

To use this functionality, your application should target the .NET Framework 3.5 or later.

See also

- String
- Binder
- DefaultBinder
- BindingFlags
- ParameterModifier
- ParameterAttributes
- CultureInfo
- ReflectionPermission

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsArrayImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the IsArray property and determines whether the Type is an array.

C#

```
protected abstract bool IsArrayImpl ();
```

Returns

Boolean

`true` if the Type is an array; otherwise, `false`.

Examples

The following example overrides the `IsArrayImpl` method in the `MyTypeDelegator` class, checks if a variable is an array, and displays the result.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegator : TypeDelegator
{
    public string myElementType = null;
    public Type myType;
    public MyTypeDelegator(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override IsArrayImpl().
    protected override bool IsArrayImpl()
    {
        // Determine whether the type is an array.
        if(myType.IsArray)
        {
```

```
            myElementType = "array";
            return true;
        }
        // Return false if the type is not an array.
        return false;
    }
}
public class Type_IsArrayImpl
{
    public static void Main()
    {
        try
        {
            int myInt = 0 ;
            // Create an instance of an array element.
            int[] myArray = new int[5];
            MyTypeDelegator myType = new MyTypeDelegator(myArray.GetType());
            Console.WriteLine("\nDetermine whether the variable is an
array.\n");
            // Determine whether myType is an array type.
            if( myType.IsArray)
                Console.WriteLine("The type of myArray is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myArray is not an array.");
            myType = new MyTypeDelegator(myInt.GetType());

            // Determine whether myType is an array type.
            if( myType.IsArray)
                Console.WriteLine("The type of myInt is {0}.",
myType.myElementType);
            else
                Console.WriteLine("myInt is not an array.");
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message );
        }
    }
}
```

Remarks

An instance of the Array class must return `false` because it is an object, not an array.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsArray

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsAssignableFrom(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether an instance of a specified type `c` can be assigned to a variable of the current type.

```
C#
public virtual bool IsAssignableFrom (Type? c);
```

Parameters

c Type

The type to compare with the current type.

Returns

Boolean

`true` if any of the following conditions is true:

- `c` and the current instance represent the same type.
- `c` is derived either directly or indirectly from the current instance. `c` is derived directly from the current instance if it inherits from the current instance; `c` is derived indirectly from the current instance if it inherits from a succession of one or more classes that inherit from the current instance.
- The current instance is an interface that `c` implements.
- `c` is a generic type parameter, and the current instance represents one of the constraints of `c`.
- `c` represents a value type, and the current instance represents `Nullable<c>` (`Nullable(Of c)` in Visual Basic).

`false` if none of these conditions are true, or if `c` is `null`.

Examples

The following example demonstrates the `IsAssignableFrom` method using defined classes, integer arrays, and generics.

C#

```
using System;
using System.Collections.Generic;
class Program
{
    public static void Main()
    {
            // Demonstrate classes:
            Console.WriteLine("Defined Classes:");
            Room room1 = new Room();
            Kitchen kitchen1 = new Kitchen();
            Bedroom bedroom1 = new Bedroom();
            Guestroom guestroom1 = new Guestroom();
            MasterBedroom masterbedroom1 = new MasterBedroom();

            Type room1Type = room1.GetType();
            Type kitchen1Type = kitchen1.GetType();
            Type bedroom1Type = bedroom1.GetType();
            Type guestroom1Type = guestroom1.GetType();
            Type masterbedroom1Type = masterbedroom1.GetType();

            Console.WriteLine("room assignable from kitchen: {0}",
room1Type.IsAssignableFrom(kitchen1Type));
            Console.WriteLine("bedroom assignable from guestroom: {0}",
bedroom1Type.IsAssignableFrom(guestroom1Type));
            Console.WriteLine("kitchen assignable from masterbedroom: {0}",
kitchen1Type.IsAssignableFrom(masterbedroom1Type));

            // Demonstrate arrays:
            Console.WriteLine();
            Console.WriteLine("Integer arrays:");

            int[] array2 = new int[2];
            int[] array10 = new int[10];
            int[,] array22 = new int[2, 2];
            int[,] array24 = new int[2, 4];

            Type array2Type = array2.GetType();
            Type array10Type = array10.GetType();
            Type array22Type = array22.GetType();
            Type array24Type = array24.GetType();

            Console.WriteLine("int[2] assignable from int[10]: {0}",
array2Type.IsAssignableFrom(array10Type));
```

```
            Console.WriteLine("int[2] assignable from int[2,4]: {0}",
array2Type.IsAssignableFrom(array24Type));
            Console.WriteLine("int[2,4] assignable from int[2,2]: {0}",
array24Type.IsAssignableFrom(array22Type));

            // Demonstrate generics:
            Console.WriteLine();
            Console.WriteLine("Generics:");

            // Note that "int?[]" is the same as "Nullable<int>[]"
            int?[] arrayNull = new int?[10];
            List<int> genIntList = new List<int>();
            List<Type> genTList = new List<Type>();

            Type arrayNullType = arrayNull.GetType();
            Type genIntListType = genIntList.GetType();
            Type genTListType = genTList.GetType();

            Console.WriteLine("int[10] assignable from int?[10]: {0}",
array10Type.IsAssignableFrom(arrayNullType));
            Console.WriteLine("List<int> assignable from List<Type>: {0}",
genIntListType.IsAssignableFrom(genTListType));
            Console.WriteLine("List<Type> assignable from List<int>: {0}",
genTListType.IsAssignableFrom(genIntListType));

            Console.ReadLine();
    }
}
class Room
{
}

class Kitchen : Room
{
}

class Bedroom : Room
{
}

class Guestroom : Bedroom
{
}

class MasterBedroom : Bedroom
{
}

//This code example produces the following output:
//
// Defined Classes:
// room assignable from kitchen: True
// bedroom assignable from guestroom: True
// kitchen assignable from masterbedroom: False
//
```

```
// Integer arrays:
// int[2] assignable from int[10]: True
// int[2] assignable from int[2,4]: False
// int[2,4] assignable from int[2,2]: True
//
// Generics:
// int[10] assignable from int?[10]: False
// List<int> assignable from List<Type>: False
// List<Type> assignable from List<int>: False
```

In the following example, the current instance is a Type object that represents the Stream class. `GenericWithConstraint` is a generic type whose generic type parameter must be of type Stream. Passing its generic type parameter to the `IsAssignableFrom` method indicates that an instance of the generic type parameter can be assigned to an Stream object.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      Type t = typeof(Stream);
      Type genericT = typeof(GenericWithConstraint<>);
      Type genericParam = genericT.GetGenericArguments()[0];
      Console.WriteLine(t.IsAssignableFrom(genericParam));
      // Displays True.
   }
}

public class GenericWithConstraint<T> where T : Stream
{}
```

Remarks

The IsAssignableFrom method can be used to determine whether an instance of `c` can be assigned to an instance of the current type, The method is most useful when you are handling objects whose types are not known at design time and allows for conditional assignment, as the following example shows.

C#

```
using System;
using System.Collections;
```

```
public class Example
{
   public static void Main()
   {
      Type t = typeof(IEnumerable);
      Type c = typeof(Array);

      IEnumerable instanceOfT;
      int[] instanceOfC = { 1, 2, 3, 4 };
      if (t.IsAssignableFrom(c))
         instanceOfT = instanceOfC;
  }
}
```

This method thus ensures that a line of code like the following will execute at runtime without throwing an InvalidCastException exception or a similar exception:

C#

```
instanceOfT = instanceOfC;
```

This method can be overridden by a derived class.

> **Note**
>
> A generic type definition is not assignable from a closed constructed type. That is, you cannot assign the closed constructed type `MyGenericList<int>` (`MyGenericList(Of Integer)` in Visual Basic) to a variable of type `MyGenericList<T>`.

If the `c` parameter is of type TypeBuilder, the result is based on the type that is to be built. The following code example demonstrates this using a built type named `B`.

C#

```
using System;
using System.Reflection;
using System.Reflection.Emit;

public class A
{}

public class Example
{
   public static void Main()
   {
      AppDomain domain = AppDomain.CurrentDomain;
```

```
        AssemblyName assemName = new AssemblyName();
        assemName.Name = "TempAssembly";

        // Define a dynamic assembly in the current application domain.
        AssemblyBuilder assemBuilder = domain.DefineDynamicAssembly(assemName,
                                                AssemblyBuilderAccess.Run);

        // Define a dynamic module in this assembly.
        ModuleBuilder moduleBuilder =
assemBuilder.DefineDynamicModule("TempModule");

        TypeBuilder b1 = moduleBuilder.DefineType("B", TypeAttributes.Public,
typeof(A));
        Console.WriteLine(typeof(A).IsAssignableFrom(b1));
    }
}
// The example displays the following output:
//         True
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsAssignableTo(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the current type can be assigned to a variable of the specified `targetType`.

```
C#
public bool IsAssignableTo (Type? targetType);
```

Parameters

`targetType` Type

The type to compare with the current type.

Returns

Boolean

`true` if any of the following conditions is true:

- The current instance and `targetType` represent the same type.
- The current type is derived either directly or indirectly from `targetType`. The current type is derived directly from `targetType` if it inherits from `targetType`; the current type is derived indirectly from `targetType` if it inherits from a succession of one or more classes that inherit from `targetType`.
- `targetType` is an interface that the current type implements.
- The current type is a generic type parameter, and `targetType` represents one of the constraints of the current type.
- The current type represents a value type, and `targetType` represents `Nullable<c>` (`Nullable(Of c)` in Visual Basic).

`false` if none of these conditions are true, or if `targetType` is `null`.

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsByRefImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the IsByRef property and determines whether the Type is passed by reference.

C#

```
protected abstract bool IsByRefImpl ();
```

Returns

Boolean

`true` if the Type is passed by reference; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsByRef

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.IsCOMObjectImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the IsCOMObject property and determines whether the Type is a COM object.

```
C#
protected abstract bool IsCOMObjectImpl ();
```

Returns

Boolean

`true` if the Type is a COM object; otherwise, `false`.

Remarks

This method returns `false` for COM interfaces because they are not objects. COM interfaces can be implemented by Microsoft .NET Framework objects.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsCOMObject

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsContextfulImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Implements the IsContextful property and determines whether the Type can be hosted in a context.

C#

```
protected virtual bool IsContextfulImpl ();
```

Returns

Boolean

`true` if the Type can be hosted in a context; otherwise, `false`.

Examples

The following example demonstrates a use of the `IsContextfulImpl` method.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegatorClass : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;

    public MyTypeDelegatorClass(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override IsContextfulImpl.
    protected override bool IsContextfulImpl()
    {
        // Check whether the type is contextful.
        if(myType.IsContextful)
        {
```

```
            myElementType = " is contextful ";
            return true;
        }
        return false;
    }
}
public class MyTypeDemoClass
{
    public static void Main()
    {
        try
        {
            MyTypeDelegatorClass myType;
            Console.WriteLine ("Check whether MyContextBoundClass can be
hosted in a context.");
            // Check whether MyContextBoundClass is contextful.
            myType = new MyTypeDelegatorClass(typeof(MyContextBoundClass));
            if( myType.IsContextful)
            {
                Console.WriteLine(typeof(MyContextBoundClass) + " can be
hosted in a context.");
            }
            else
            {
                Console.WriteLine(typeof(MyContextBoundClass) + " cannot be
hosted in a context.");
            }
            // Check whether the int type is contextful.
            myType = new MyTypeDelegatorClass(typeof(MyTypeDemoClass));
            Console.WriteLine ("\nCheck whether MyTypeDemoClass can be
hosted in a context.");
            if( myType.IsContextful)
            {
                Console.WriteLine(typeof(MyTypeDemoClass) + " can be hosted
in a context.");
            }
            else
            {
                Console.WriteLine(typeof(MyTypeDemoClass) + " cannot be
hosted in a context.");
            }
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
        }
    }
}
// This class demonstrates IsContextfulImpl.
public class MyContextBoundClass : ContextBoundObject
{
    public string myString = "This class is used to demonstrate members of
the Type class.";
}
```

Remarks

This method can be overridden by a derived class.

A context intercepts calls to the class members and enforce policies that are applied to the class, such as synchronization.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsContextful

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsEnumDefined(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the specified value exists in the current enumeration type.

```
C#
public virtual bool IsEnumDefined (object value);
```

Parameters

`value` Object

The value to be tested.

Returns

Boolean

`true` if the specified value is a member of the current enumeration type; otherwise, `false`.

Exceptions

ArgumentException

The current type is not an enumeration.

ArgumentNullException

`value` is `null`.

InvalidOperationException

`value` is of a type that cannot be the underlying type of an enumeration.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsEquivalentTo(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two COM types have the same identity and are eligible for type equivalence.

```
C#
public virtual bool IsEquivalentTo (Type? other);
```

Parameters

`other` Type

The COM type that is tested for equivalence with the current type.

Returns

Boolean

`true` if the COM types are equivalent; otherwise, `false`. This method also returns `false` if one type is in an assembly that is loaded for execution, and the other is in an assembly that is loaded into the reflection-only context.

Remarks

Beginning with the .NET Framework 4, the common language runtime supports the embedding of type information for COM types directly into managed assemblies, instead of requiring the managed assemblies to obtain type information for COM types from interop assemblies. Because the embedded type information includes only the types and members that are actually used by a managed assembly, two managed assemblies might have very different views of the same COM type. Each managed assembly has a different Type object to represent its view of the COM type. The common language runtime supports type equivalence between these different views for interfaces, structures, enumerations, and delegates.

Type equivalence means that a COM object that is passed from one managed assembly to another can be cast to the appropriate managed type in the receiving assembly. The IsEquivalentTo method enables an assembly to determine that a COM object obtained from another assembly has the same COM identity as one of the first assembly's own embedded interop types, and thus can be cast to that type.

For more information, see Type Equivalence and Embedded Interop Types.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsInstanceOfType(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified object is an instance of the current Type.

C#

```
public virtual bool IsInstanceOfType (object? o);
```

Parameters

o Object

The object to compare with the current type.

Returns

Boolean

`true` if the current `Type` is in the inheritance hierarchy of the object represented by `o`, or if the current `Type` is an interface that `o` implements. `false` if neither of these conditions is the case, if `o` is `null`, or if the current `Type` is an open generic type (that is, ContainsGenericParameters returns `true`).

Examples

The following example demonstrates the use of the `IsInstanceOfType` method.

C#

```
using System;

public interface IExample {}

public class BaseClass : IExample {}

public class DerivedClass : BaseClass {}
```

```
public class Example
{
    public static void Main()
    {
        var interfaceType = typeof(IExample);
        var base1 = new BaseClass();
        var base1Type = base1.GetType();
        var derived1 = new DerivedClass();
        var derived1Type = derived1.GetType();
        int[] arr = new int[11];

        Console.WriteLine("Is int[] an instance of the Array class? {0}.",
                           typeof(Array).IsInstanceOfType(arr));
        Console.WriteLine("Is base1 an instance of BaseClass? {0}.",
                          base1Type.IsInstanceOfType(base1));
        Console.WriteLine("Is derived1 an instance of BaseClass? {0}.",
                          base1Type.IsInstanceOfType(derived1));
        Console.WriteLine("Is base1 an instance of IExample? {0}.",
                          interfaceType.IsInstanceOfType(base1));
        Console.WriteLine("Is derived1 an instance of IExample? {0}.",
                          interfaceType.IsInstanceOfType(derived1));
    }
}
// The example displays the following output:
//    Is int[] an instance of the Array class? True.
//    Is base1 an instance of BaseClass? True.
//    Is derived1 an instance of BaseClass? True.
//    Is base1 an instance of IExample? True.
//    Is derived1 an instance of IExample? True.
```

Remarks

This method can be overridden by a derived class.

> ⓘ **Note**
>
> A constructed type is not an instance of its generic type definition. That is, `MyGenericList<int>` (`MyGenericList(Of Integer)` in Visual Basic) is not an instance of `MyGenericList<T>` (`MyGenericList(Of T)` in Visual Basic).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsMarshalByRefImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Implements the IsMarshalByRef property and determines whether the Type is marshaled by reference.

C#

```
protected virtual bool IsMarshalByRefImpl ();
```

Returns

Boolean

`true` if the Type is marshaled by reference; otherwise, `false`.

Examples

The following example determines whether the given type is marshaled by reference and displays the result.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegatorClass : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;
    public MyTypeDelegatorClass(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override IsMarshalByRefImpl.
    protected override bool IsMarshalByRefImpl()
    {
        // Determine whether the type is marshalled by reference.
        if(myType.IsMarshalByRef)
        {
```

```
            myElementType = " marshalled by reference";
            return true;
        }
        return false;
    }
}
public class MyTypeDemoClass
{
    public static void Main()
    {
        try
        {
            MyTypeDelegatorClass myType;
            Console.WriteLine ("Determine whether MyContextBoundClass is
marshalled by reference.");
            // Determine whether MyContextBoundClass type is marshalled by
reference.
            myType = new MyTypeDelegatorClass(typeof(MyContextBoundClass));
            if( myType.IsMarshalByRef )
            {
                Console.WriteLine(typeof(MyContextBoundClass) + " is
marshalled by reference.");
            }
            else
            {
                Console.WriteLine(typeof(MyContextBoundClass) + " is not
marshalled by reference.");
            }

            // Determine whether int type is marshalled by reference.
            myType = new MyTypeDelegatorClass(typeof(int));
            Console.WriteLine ("\nDetermine whether int is marshalled by
reference.");
            if( myType.IsMarshalByRef)
            {
                Console.WriteLine(typeof(int) + " is marshalled by
reference.");
            }
            else
            {
                Console.WriteLine(typeof(int) + " is not marshalled by
reference.");
            }
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
        }
    }
}
// This class is used to demonstrate the IsMarshalByRefImpl method.
public class MyContextBoundClass : ContextBoundObject
{
    public string myString = "This class is used to demonstrate members of
```

```
the Type class.";
}
```

Remarks

This method can be overridden by a derived class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsMarshalByRef

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsPointerImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the IsPointer property and determines whether the Type is a pointer.

```
C#
protected abstract bool IsPointerImpl ();
```

Returns

Boolean

`true` if the Type is a pointer; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IsPointer

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Type.IsPrimitiveImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

When overridden in a derived class, implements the IsPrimitive property and determines whether the Type is one of the primitive types.

C#

```
protected abstract bool IsPrimitiveImpl ();
```

Returns

Boolean

`true` if the Type is one of the primitive types; otherwise, `false`.

Examples

The following example determines whether the given type is a primitive type and displays the result.

C#

```
using System;
using System.Reflection;
public class MyTypeDelegatorClass : TypeDelegator
{
    public string myElementType = null;
    private Type myType = null ;
    public MyTypeDelegatorClass(Type myType) : base(myType)
    {
        this.myType = myType;
    }
    // Override the IsPrimitiveImpl.
    protected override bool IsPrimitiveImpl()
    {
        // Determine whether the type is a primitive type.
        if(myType.IsPrimitive)
        {
```

```
            myElementType = "primitive";
            return true;
        }
        return false;
    }
}
public class MyTypeDemoClass
{
    public static void Main()
    {
        try
        {
            Console.WriteLine ("Determine whether int is a primitive
type.");
            MyTypeDelegatorClass myType;
            myType = new MyTypeDelegatorClass(typeof(int));
            // Determine whether int is a primitive type.
            if( myType.IsPrimitive)
            {
                Console.WriteLine(typeof(int) + " is a primitive type.");
            }
            else
            {
                Console.WriteLine(typeof(int) + " is not a primitive
type.");
            }
            Console.WriteLine ("\nDetermine whether string is a primitive
type.");
            myType = new MyTypeDelegatorClass(typeof(string));
            // Determine if string is a primitive type.
            if( myType.IsPrimitive)
            {
                Console.WriteLine(typeof(string) + " is a primitive type.");
            }
            else
            {
                Console.WriteLine(typeof(string) + " is not a primitive
type.");
            }
        }
        catch( Exception e )
        {
            Console.WriteLine("Exception: {0}", e.Message);
        }
    }
}
```

Remarks

The primitive types are Boolean, Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64, Char, Double, and Single.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Boolean
- Byte
- SByte
- Int16
- UInt16
- Int32
- UInt32
- Int64
- UInt64
- Char
- Double
- Single
- IsPrimitive

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.IsSubclassOf(Type) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the current Type derives from the specified Type.

```
C#
public virtual bool IsSubclassOf (Type c);
```

Parameters

c Type

The type to compare with the current type.

Returns

Boolean

`true` if the current `Type` derives from `c`; otherwise, `false`. This method also returns `false` if `c` and the current `Type` are equal.

Exceptions

ArgumentNullException

`c` is `null`.

Examples

The following example creates a class named `Class1` and a derived class named `DerivedC1`. It calls the IsSubclassOf method to show that `DerivedC1` is a subclass of `Class1`.

```
C#
```

```
using System;

public class Class1 { }
public class DerivedC1 : Class1 { }

class IsSubclassTest
{
   public static void Main()
   {
      Console.WriteLine("DerivedC1 subclass of Class1: {0}",
                        typeof(DerivedC1).IsSubclassOf(typeof(Class1)));
   }
}
// The example displays the following output:
//        DerivedC1 subclass of Class1: True
```

Remarks

You can call the IsSubclassOf method to determine any of the following:

- Whether one class derives from another.
- Whether a type derives from ValueType. However, the IsValueType is a more efficient way to determine whether a type is a value type.
- Whether a type derives from Enum. However, the IsEnum method is a more efficient way to determine whether a type is an enumeration.
- Whether a type is a delegate, that is, whether it derives from either Delegate or MulticastDelegate.

The IsSubclassOf method cannot be used to determine whether an interface derives from another interface, or whether a class implements an interface. Use the IsAssignableFrom method for that purpose, as the following example shows.

C#

```
using System;

public interface IInterface
{
   void Display();
}

public class Implementation : IInterface
{
   public void Display()
   {
```

```
        Console.WriteLine("The implementation...");
    }
}

public class Example
{
    public static void Main()
    {
        Console.WriteLine("Implementation is a subclass of IInterface:   {0}",

typeof(Implementation).IsSubclassOf(typeof(IInterface)));
        Console.WriteLine("IInterface is assignable from Implementation: {0}",

typeof(IInterface).IsAssignableFrom(typeof(Implementation)));
    }
}
// The example displays the following output:
//       Implementation is a subclass of IInterface:   False
//       IInterface is assignable from Implementation: True
```

If the current Type represents a type parameter in the definition of a generic type or generic method, it derives from its class constraint or from System.Object if it has no class constraint.

> **Note**
>
> Except when used with interfaces, **IsSubclassOf** is the converse of **IsAssignableFrom**. That is, if `t1.IsSubclassOf(t2)` is `true`, then `t2.IsAssignableFrom(t1)` is also `true`.

This method can be overridden by a derived class.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- BaseType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.IsValueTypeImpl Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Implements the IsValueType property and determines whether the Type is a value type; that is, not a class or an interface.

```
C#
protected virtual bool IsValueTypeImpl ();
```

Returns

Boolean

`true` if the Type is a value type; otherwise, `false`.

Remarks

This method is provided to enable the implementation of alternate type systems. It is not generally used in application code.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeAttributes
- IsClass

- IsInterface
- ValueType
- IsValueType

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.MakeArrayType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a Type object that represents an array of the current type.

Overloads

Expand table

MakeArrayType()	Returns a Type object representing a one-dimensional array of the current type, with a lower bound of zero.
MakeArrayType(Int32)	Returns a Type object representing an array of the current type, with the specified number of dimensions.

Examples

The following code example creates array, `ref` (`ByRef` in Visual Basic), and pointer types for the `Test` class.

C#

```
using System;
using System.Reflection;

public class Example
{
    public static void Main()
    {
        // Create a Type object that represents a one-dimensional
        // array of Example objects.
        Type t = typeof(Example).MakeArrayType();
        Console.WriteLine("\r\nArray of Example: {0}", t);

        // Create a Type object that represents a two-dimensional
        // array of Example objects.
        t = typeof(Example).MakeArrayType(2);
        Console.WriteLine("\r\nTwo-dimensional array of Example: {0}", t);
```

```
        // Demonstrate an exception when an invalid array rank is
        // specified.
        try
        {
            t = typeof(Example).MakeArrayType(-1);
        }
        catch(Exception ex)
        {
            Console.WriteLine("\r\n{0}", ex);
        }

        // Create a Type object that represents a ByRef parameter
        // of type Example.
        t = typeof(Example).MakeByRefType();
        Console.WriteLine("\r\nByRef Example: {0}", t);

        // Get a Type object representing the Example class, a
        // MethodInfo representing the "Test" method, a ParameterInfo
        // representing the parameter of type Example, and finally
        // a Type object representing the type of this ByRef parameter.
        // Compare this Type object with the Type object created using
        // MakeByRefType.
        Type t2 = typeof(Example);
        MethodInfo mi = t2.GetMethod("Test");
        ParameterInfo pi = mi.GetParameters()[0];
        Type pt = pi.ParameterType;
        Console.WriteLine("Are the ByRef types equal? {0}", (t == pt));

        // Create a Type object that represents a pointer to an
        // Example object.
        t = typeof(Example).MakePointerType();
        Console.WriteLine("\r\nPointer to Example: {0}", t);
    }

    // A sample method with a ByRef parameter.
    //
    public void Test(ref Example e)
    {
    }
}

/* This example produces output similar to the following:

Array of Example: Example[]

Two-dimensional array of Example: Example[,]

System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at System.RuntimeType.MakeArrayType(Int32 rank) in
c:\vbl\ndp\clr\src\BCL\System\RtType.cs:line 2999
   at Example.Main()

ByRef Example: Example&
Are the ByRef types equal? True
```

```
Pointer to Example: Example*

 */
```

MakeArrayType()

Returns a Type object representing a one-dimensional array of the current type, with a lower bound of zero.

```
C#

public virtual Type MakeArrayType ();
```

Returns

Type

A Type object representing a one-dimensional array of the current type, with a lower bound of zero.

Exceptions

NotSupportedException

The invoked method is not supported in the base class. Derived classes must provide an implementation.

TypeLoadException

The current type is TypedReference.

-or-

The current type is a `ByRef` type. That is, IsByRef returns `true`.

Remarks

The MakeArrayType method provides a way to generate array types whose element types are computed at run time.

Note The common language runtime makes a distinction between vectors (that is, one-dimensional arrays that are always zero-based) and multidimensional arrays. A vector, which always has only one dimension, is not the same as a multidimensional array that happens to have only one dimension. This method overload can only be

used to create vector types, and it is the only way to create a vector type. Use the MakeArrayType(Int32) method overload to create multidimensional array types.

See also

- MakeByRefType()
- MakePointerType()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

MakeArrayType(Int32)

Returns a Type object representing an array of the current type, with the specified number of dimensions.

C#

```
public virtual Type MakeArrayType (int rank);
```

Parameters

`rank` Int32

The number of dimensions for the array. This number must be less than or equal to 32.

Returns

Type

An object representing an array of the current type, with the specified number of dimensions.

Exceptions

IndexOutOfRangeException

`rank` is invalid. For example, 0 or negative.

NotSupportedException

The invoked method is not supported in the base class.

TypeLoadException

The current type is TypedReference.

-or-

The current type is a `ByRef` type. That is, IsByRef returns `true`.

-or-

`rank` is greater than 32.

Remarks

The MakeArrayType method provides a way to generate array types whose element types are computed at run time.

> **Note**
>
> The common language runtime makes a distinction between vectors (that is, one-dimensional arrays that are always zero-based) and multidimensional arrays. A vector, which always has only one dimension, is not the same as a multidimensional array that happens to have only one dimension. You cannot use this method overload to create a vector type; if `rank` is 1, this method overload returns a multidimensional array type that happens to have one dimension. Use the **MakeArrayType()** method overload to create vector types.

See also

- MakeByRefType()
- MakePointerType()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.MakeByRefType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a Type object that represents the current type when passed as a `ref` parameter (`ByRef` parameter in Visual Basic).

C#

```
public virtual Type MakeByRefType ();
```

Returns

Type

A Type object that represents the current type when passed as a `ref` parameter (`ByRef` parameter in Visual Basic).

Exceptions

NotSupportedException

The invoked method is not supported in the base class.

TypeLoadException

The current type is TypedReference.

-or-

The current type is a `ByRef` type. That is, IsByRef returns `true`.

Examples

The following code example creates array, `ref` (`ByRef` in Visual Basic), and pointer types for the `Test` class.

C#

```
using System;
using System.Reflection;

public class Example
{
    public static void Main()
    {
        // Create a Type object that represents a one-dimensional
        // array of Example objects.
        Type t = typeof(Example).MakeArrayType();
        Console.WriteLine("\r\nArray of Example: {0}", t);

        // Create a Type object that represents a two-dimensional
        // array of Example objects.
        t = typeof(Example).MakeArrayType(2);
        Console.WriteLine("\r\nTwo-dimensional array of Example: {0}", t);

        // Demonstrate an exception when an invalid array rank is
        // specified.
        try
        {
            t = typeof(Example).MakeArrayType(-1);
        }
        catch(Exception ex)
        {
            Console.WriteLine("\r\n{0}", ex);
        }

        // Create a Type object that represents a ByRef parameter
        // of type Example.
        t = typeof(Example).MakeByRefType();
        Console.WriteLine("\r\nByRef Example: {0}", t);

        // Get a Type object representing the Example class, a
        // MethodInfo representing the "Test" method, a ParameterInfo
        // representing the parameter of type Example, and finally
        // a Type object representing the type of this ByRef parameter.
        // Compare this Type object with the Type object created using
        // MakeByRefType.
        Type t2 = typeof(Example);
        MethodInfo mi = t2.GetMethod("Test");
        ParameterInfo pi = mi.GetParameters()[0];
        Type pt = pi.ParameterType;
        Console.WriteLine("Are the ByRef types equal? {0}", (t == pt));

        // Create a Type object that represents a pointer to an
        // Example object.
        t = typeof(Example).MakePointerType();
        Console.WriteLine("\r\nPointer to Example: {0}", t);
    }

    // A sample method with a ByRef parameter.
    //
    public void Test(ref Example e)
```

```
    {
    }
}

/* This example produces output similar to the following:

Array of Example: Example[]

Two-dimensional array of Example: Example[,]

System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at System.RuntimeType.MakeArrayType(Int32 rank) in
c:\vbl\ndp\clr\src\BCL\System\RtType.cs:line 2999
   at Example.Main()

ByRef Example: Example&
Are the ByRef types equal? True

Pointer to Example: Example*

 */
```

Remarks

The MakeByRefType method provides a way to generate `ref` types (`ByRef` in Visual Basic) for parameter lists.

Using the syntax of Microsoft intermediate language (MSIL), if the current Type object represents Int32, this method returns a Type object representing `Int32&`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MakeArrayType()
- MakePointerType()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.MakeGenericMethodParameter(Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a signature type object that can be passed into the `Type[]` array parameter of a GetMethod method to represent a generic parameter reference.

```
C#
public static Type MakeGenericMethodParameter (int position);
```

Parameters

`position` Int32

The typed parameter position.

Returns

Type

A signature type object that can be passed into the `Type[]` array parameter of a GetMethod method to represent a generic parameter reference.

Exceptions

ArgumentOutOfRangeException

`position` is negative.

Applies to

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.MakeGenericSignatureType(Type, Type[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a generic signature type, which allows third party reimplementations of Reflection to fully support the use of signature types in querying type members.

C#

```
public static Type MakeGenericSignatureType (Type genericTypeDefinition,
params Type[] typeArguments);
```

Parameters

`genericTypeDefinition` Type

The generic type definition.

`typeArguments` Type[]

An array of type arguments.

Returns

Type

A generic signature type.

Applies to

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.MakeGenericType(Type[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Substitutes the elements of an array of types for the type parameters of the current generic type definition and returns a Type object representing the resulting constructed type.

```
C#
public virtual Type MakeGenericType (params Type[] typeArguments);
```

Parameters

typeArguments Type[]

An array of types to be substituted for the type parameters of the current generic type.

Returns

Type

A Type representing the constructed type formed by substituting the elements of `typeArguments` for the type parameters of the current generic type.

Exceptions

InvalidOperationException

The current type does not represent a generic type definition. That is, IsGenericTypeDefinition returns `false`.

ArgumentNullException

`typeArguments` is `null`.

-or-

Any element of `typeArguments` is `null`.

ArgumentException

The number of elements in `typeArguments` is not the same as the number of type parameters in the current generic type definition.

-or-

Any element of `typeArguments` does not satisfy the constraints specified for the corresponding type parameter of the current generic type.

-or-

`typeArguments` contains an element that is a pointer type (IsPointer returns `true`), a by-ref type (IsByRef returns `true`), or Void.

NotSupportedException

The invoked method is not supported in the base class. Derived classes must provide an implementation.

Examples

The following example uses the MakeGenericType method to create a constructed type from the generic type definition for the Dictionary<TKey,TValue> type. The constructed type represents a Dictionary<TKey,TValue> of `Test` objects with string keys.

C#

```
using System;
using System.Reflection;
using System.Collections.Generic;

public class Test
{
    public static void Main()
    {
        Console.WriteLine("\r\n--- Create a constructed type from the
generic Dictionary type.");

        // Create a type object representing the generic Dictionary
        // type, by omitting the type arguments (but keeping the
        // comma that separates them, so the compiler can infer the
        // number of type parameters).
        Type generic = typeof(Dictionary<,>);
        DisplayTypeInfo(generic);

        // Create an array of types to substitute for the type
        // parameters of Dictionary. The key is of type string, and
        // the type to be contained in the Dictionary is Test.
```

```
        Type[] typeArgs = { typeof(string), typeof(Test) };

        // Create a Type object representing the constructed generic
        // type.
        Type constructed = generic.MakeGenericType(typeArgs);
        DisplayTypeInfo(constructed);

        // Compare the type objects obtained above to type objects
        // obtained using typeof() and GetGenericTypeDefinition().
        Console.WriteLine("\r\n--- Compare types obtained by different
methods:");

        Type t = typeof(Dictionary<String, Test>);
        Console.WriteLine("\tAre the constructed types equal? {0}", t ==
constructed);
        Console.WriteLine("\tAre the generic types equal? {0}",
            t.GetGenericTypeDefinition() == generic);
    }

    private static void DisplayTypeInfo(Type t)
    {
        Console.WriteLine("\r\n{0}", t);

        Console.WriteLine("\tIs this a generic type definition? {0}",
            t.IsGenericTypeDefinition);

        Console.WriteLine("\tIs it a generic type? {0}",
            t.IsGenericType);

        Type[] typeArguments = t.GetGenericArguments();
        Console.WriteLine("\tList type arguments ({0}):",
typeArguments.Length);
        foreach (Type tParam in typeArguments)
        {
            Console.WriteLine("\t\t{0}", tParam);
        }
    }
}

/* This example produces the following output:

--- Create a constructed type from the generic Dictionary type.

System.Collections.Generic.Dictionary`2[TKey,TValue]
        Is this a generic type definition? True
        Is it a generic type? True
        List type arguments (2):
                TKey
                TValue

System.Collections.Generic.Dictionary`2[System.String, Test]
        Is this a generic type definition? False
        Is it a generic type? True
        List type arguments (2):
                System.String
```

```
                 Test

--- Compare types obtained by different methods:
        Are the constructed types equal? True
        Are the generic types equal? True
 */
```

Remarks

For more information about this API, see Supplemental API remarks for Type.MakeGenericType.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- IsGenericTypeDefinition
- GetGenericTypeDefinition()
- Reflection and Generic Types
- How to: Examine and Instantiate Generic Types with Reflection

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.MakePointerType Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a Type object that represents a pointer to the current type.

```
C#

public virtual Type MakePointerType ();
```

Returns

Type

A Type object that represents a pointer to the current type.

Exceptions

NotSupportedException

The invoked method is not supported in the base class.

TypeLoadException

The current type is TypedReference.

-or-

The current type is a `ByRef` type. That is, IsByRef returns `true`.

Examples

The following code example creates array, `ref` (`ByRef` in Visual Basic), and pointer types for the `Test` class.

```
C#

using System;
using System.Reflection;
```

```
public class Example
{
    public static void Main()
    {
        // Create a Type object that represents a one-dimensional
        // array of Example objects.
        Type t = typeof(Example).MakeArrayType();
        Console.WriteLine("\r\nArray of Example: {0}", t);

        // Create a Type object that represents a two-dimensional
        // array of Example objects.
        t = typeof(Example).MakeArrayType(2);
        Console.WriteLine("\r\nTwo-dimensional array of Example: {0}", t);

        // Demonstrate an exception when an invalid array rank is
        // specified.
        try
        {
            t = typeof(Example).MakeArrayType(-1);
        }
        catch(Exception ex)
        {
            Console.WriteLine("\r\n{0}", ex);
        }

        // Create a Type object that represents a ByRef parameter
        // of type Example.
        t = typeof(Example).MakeByRefType();
        Console.WriteLine("\r\nByRef Example: {0}", t);

        // Get a Type object representing the Example class, a
        // MethodInfo representing the "Test" method, a ParameterInfo
        // representing the parameter of type Example, and finally
        // a Type object representing the type of this ByRef parameter.
        // Compare this Type object with the Type object created using
        // MakeByRefType.
        Type t2 = typeof(Example);
        MethodInfo mi = t2.GetMethod("Test");
        ParameterInfo pi = mi.GetParameters()[0];
        Type pt = pi.ParameterType;
        Console.WriteLine("Are the ByRef types equal? {0}", (t == pt));

        // Create a Type object that represents a pointer to an
        // Example object.
        t = typeof(Example).MakePointerType();
        Console.WriteLine("\r\nPointer to Example: {0}", t);
    }

    // A sample method with a ByRef parameter.
    //
    public void Test(ref Example e)
    {
    }
}
```

```
/* This example produces output similar to the following:

Array of Example: Example[]

Two-dimensional array of Example: Example[,]

System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at System.RuntimeType.MakeArrayType(Int32 rank) in
c:\vbl\ndp\clr\src\BCL\System\RtType.cs:line 2999
   at Example.Main()

ByRef Example: Example&
Are the ByRef types equal? True

Pointer to Example: Example*

 */
```

Remarks

The MakePointerType method provides a way to generate pointer types for parameter lists.

Using the syntax of Microsoft intermediate language (MSIL), if the current Type object represents Int32, this method returns a Type object representing `Int32*`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MakeByRefType()
- MakeArrayType()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.ReflectionOnlyGetType(String, Boolean, Boolean) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> ReflectionOnly loading is not supported and throws PlatformNotSupportedException.

Gets the Type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found. The type is loaded for reflection only, not for execution.

C#

```
[System.Obsolete("ReflectionOnly loading is not supported and throws PlatformNotSupportedException.", DiagnosticId="SYSLIB0018", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public static Type? ReflectionOnlyGetType (string typeName, bool throwIfNotFound, bool ignoreCase);
```

Parameters

`typeName` String

The assembly-qualified name of the Type to get.

`throwIfNotFound` Boolean

`true` to throw a TypeLoadException if the type cannot be found; `false` to return `null` if the type cannot be found. Specifying `false` also suppresses some other exception conditions, but not all of them. See the Exceptions section.

`ignoreCase` Boolean

`true` to perform a case-insensitive search for `typeName`; `false` to perform a case-sensitive search for `typeName`.

Returns

Type

The type with the specified name, if found; otherwise, `null`. If the type is not found, the `throwIfNotFound` parameter specifies whether `null` is returned or an exception is thrown. In some cases, an exception is thrown regardless of the value of `throwIfNotFound`. See the Exceptions section.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`typeName` is `null`.

TargetInvocationException

A class initializer is invoked and throws an exception.

TypeLoadException

`throwIfNotFound` is `true` and the type is not found.

-or-

`throwIfNotFound` is `true` and `typeName` contains invalid characters, such as an embedded tab.

-or-

`throwIfNotFound` is `true` and `typeName` is an empty string.

-or-

`throwIfNotFound` is `true` and `typeName` represents an array type with an invalid size.

-or-

`typeName` represents an array of TypedReference objects.

ArgumentException

`typeName` does not include the assembly name.

-or-

`throwIfNotFound` is `true` and `typeName` contains invalid syntax; for example, "MyType[,*,]".

-or-

`typeName` represents a generic type that has a pointer type, a `ByRef` type, or Void as one of its type arguments.

-or-

`typeName` represents a generic type that has an incorrect number of type arguments.

-or-

`typeName` represents a generic type, and one of its type arguments does not satisfy the constraints for the corresponding type parameter.

FileNotFoundException
`throwIfNotFound` is `true` and the assembly or one of its dependencies was not found.

FileLoadException
The assembly or one of its dependencies was found, but could not be loaded.

BadImageFormatException
The assembly or one of its dependencies is not valid.

-or-

The assembly was compiled with a later version of the common language runtime than the version that is currently loaded.

PlatformNotSupportedException
.NET Core and .NET 5+ only: In all cases.

Remarks

If the assembly containing the type is not already loaded into the reflection-only context, using the ReflectionOnlyGetType method is equivalent to first loading the assembly for reflection only, using the ReflectionOnlyLoad method, and then loading

the type by calling the assembly's Assembly.GetType method. For information about assembly-qualified names, see the Type.AssemblyQualifiedName property. For additional details on specifying type names, see the GetType(String, Boolean, Boolean) method overload.

If the assembly is already loaded for execution, another copy is loaded into the reflection-only context.

The `throwIfNotFound` parameter specifies what happens when the type is not found, and also suppresses certain other exception conditions, as described in the Exceptions section. Some exceptions are thrown regardless of the value of `throwIfNotFound`. For example, if the assembly is not valid, a BadImageFormatException is thrown even if `throwIfNotFound` is `false`.

For more information about using the reflection-only context, see How to: Load Assemblies into the Reflection-Only Context.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- String
- TypeLoadException
- AssemblyQualifiedName
- GetAssembly(Type)
- GetType(String)
- AssemblyName
- Specifying Fully Qualified Type Names
- How to: Load Assemblies into the Reflection-Only Context

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a `String` representing the name of the current `Type`.

C#

```
public override string ToString ();
```

Returns

String

A String representing the name of the current Type.

Examples

This following example demonstrates a use of the Namespace and Module properties and the `ToString` method of Type.

C#

```
using System;

namespace MyNamespace
{
    class MyClass
    {
    }
}

public class Example
{
    public static void Main()
    {
         Type myType = typeof(MyNamespace.MyClass);
         Console.WriteLine("Displaying information about {0}:", myType);
         // Get the namespace of the myClass class.
         Console.WriteLine("   Namespace: {0}.", myType.Namespace);
```

```
        // Get the name of the module.
        Console.WriteLine("   Module: {0}.", myType.Module);
        // Get the fully qualified type name.
        Console.WriteLine("   Fully qualified name: {0}.",
myType.ToString());
    }
}
// The example displays the following output:
//    Displaying information about MyNamespace.MyClass:
//       Namespace: MyNamespace.
//       Module: type_tostring.exe.
//       Fully qualified name: MyNamespace.MyClass.
```

The following example compares the strings returned by the `ToString` method and the Name, FullName, and AssemblyQualifiedName properties.

C#

```
using System;
using System.Collections.Generic;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        Type t = typeof(String);
        ShowTypeInfo(t);

        t = typeof(List<>);
        ShowTypeInfo(t);

        var list = new List<String>();
        t = list.GetType();
        ShowTypeInfo(t);

        Object v = 12;
        t = v.GetType();
        ShowTypeInfo(t);

        t = typeof(IFormatProvider);
        ShowTypeInfo(t);

        IFormatProvider ifmt = NumberFormatInfo.CurrentInfo;
        t = ifmt.GetType();
        ShowTypeInfo(t);
    }

    private static void ShowTypeInfo(Type t)
    {
        Console.WriteLine($"Name: {t.Name}");
        Console.WriteLine($"Full Name: {t.FullName}");
        Console.WriteLine($"ToString:  {t}");
```

```
        Console.WriteLine($"Assembly Qualified Name:
{t.AssemblyQualifiedName}");
        Console.WriteLine();
    }
}
// The example displays output like the following:
//    Name: String
//    Full Name: System.String
//    ToString:  System.String
//    Assembly Qualified Name: System.String, mscorlib, Version=4.0.0.0,
Culture=neutr
//    al, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1
//    ToString:  System.Collections.Generic.List`1[T]
//    Assembly Qualified Name: System.Collections.Generic.List`1, mscorlib,
Version=4.
//    0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: List`1
//    Full Name: System.Collections.Generic.List`1[[System.String, mscorlib,
Version=4
//    .0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]
//    ToString:  System.Collections.Generic.List`1[System.String]
//    Assembly Qualified Name:
System.Collections.Generic.List`1[[System.String, mscor
//    lib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089]], mscorl
//    ib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: Int32
//    Full Name: System.Int32
//    ToString:  System.Int32
//    Assembly Qualified Name: System.Int32, mscorlib, Version=4.0.0.0,
Culture=neutra
//    l, PublicKeyToken=b77a5c561934e089
//
//    Name: IFormatProvider
//    Full Name: System.IFormatProvider
//    ToString:  System.IFormatProvider
//    Assembly Qualified Name: System.IFormatProvider, mscorlib,
Version=4.0.0.0, Cult
//    ure=neutral, PublicKeyToken=b77a5c561934e089
//
//    Name: NumberFormatInfo
//    Full Name: System.Globalization.NumberFormatInfo
//    ToString:  System.Globalization.NumberFormatInfo
//    Assembly Qualified Name: System.Globalization.NumberFormatInfo,
mscorlib, Versio
//    n=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Remarks

This method returns the fully qualified common language runtime namespace and name for all primitive types. For example, the C# instruction, `(long)0.Type().ToString()` returns "System.Int64" instead of merely "Int64".

If the current Type represents a generic type, the type and its type arguments are qualified by namespace and by nested type, but not by assembly. If the current Type represents a type parameter in the definition of a generic type or generic method, this method returns the unqualified name of the type parameter.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- String

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Type.Equality(Type, Type) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two Type objects are equal.

C#

```
public static bool operator == (Type? left, Type? right);
```

Parameters

`left` Type

The first object to compare.

`right` Type

The second object to compare.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Type.Inequality(Type, Type) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether two Type objects are not equal.

C#

```
public static bool operator != (Type? left, Type? right);
```

Parameters

`left` Type

The first object to compare.

`right` Type

The second object to compare.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeAccessException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when a method attempts to use a type that it does not have access to.

```
C#
public class TypeAccessException : TypeLoadException
```

Inheritance Object → Exception → SystemException → TypeLoadException → TypeAccessException

Remarks

TypeAccessException is thrown when a method attempts to cast an object to a type that is not accessible from the method. For example, an anonymously hosted dynamic method cannot access a security-critical type because the method is transparent. If the method contains a OpCodes.Castclass instruction that casts an object to a security-critical type, or to a generic type that has a security-critical type as one of its type parameters, TypeAccessException is thrown by the JIT compiler.

Similarly, TypeAccessException is thrown for a dynamic method that uses an internal type from another assembly. A method might not have access to its containing type, its return type, or one of its parameter types.

Apps compiled by using the .NET Native tool chain may throw a MissingMetadataException exception at run time. `MissingMetadataException` is an internal-only exception type derived from TypeAccessException. The exception indicates that metadata needed for the successful execution of an app is not present at run time. You should not use a `try`/`catch` block to handle the exception. Instead, you should determine what metadata is missing and modify your app's runtime directives file to ensure that it is present at run time.

Constructors

Expand table

TypeAccessException()	Initializes a new instance of the TypeAccessException class with a system-supplied message that describes the error.
TypeAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeAccessException class with serialized data.
TypeAccessException(String)	Initializes a new instance of the TypeAccessException class with a specified message that describes the error.
TypeAccessException(String, Exception)	Initializes a new instance of the TypeAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message for this exception. (Inherited from TypeLoadException)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

TypeName	Gets the fully qualified name of the type that causes the exception. (Inherited from TypeLoadException)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, method name, resource ID, and additional exception information. (Inherited from TypeLoadException)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeAccessException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TypeAccessException class.

Overloads

Expand table

TypeAccessException()	Initializes a new instance of the TypeAccessException class with a system-supplied message that describes the error.
TypeAccessException(String)	Initializes a new instance of the TypeAccessException class with a specified message that describes the error.
TypeAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeAccessException class with serialized data.
TypeAccessException(String, Exception)	Initializes a new instance of the TypeAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

TypeAccessException()

Initializes a new instance of the TypeAccessException class with a system-supplied message that describes the error.

C#

```
public TypeAccessException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempt to access the type failed." This message takes the current system culture into account.

The following table shows the initial property values for an instance of TypeAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TypeAccessException(String)

Initializes a new instance of the TypeAccessException class with a specified message that describes the error.

```
C#
```

```
public TypeAccessException (string? message);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor must ensure that this string has been localized for the current system culture.

Remarks

The following table shows the initial property values for an instance of TypeAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TypeAccessException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the TypeAccessException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected TypeAccessException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object that was transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TypeAccessException(String, Exception)

Initializes a new instance of the TypeAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public TypeAccessException (string? message, Exception? inner);
```

Parameters

`message` String

The message that describes the exception. The caller of this constructor must ensure that this string has been localized for the current system culture.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of TypeAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeCode Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the type of an object.

```
C#

public enum TypeCode
```

Inheritance Object → ValueType → Enum → TypeCode

Fields

Expand table

Boolean	3	A simple type representing Boolean values of `true` or `false`.
Byte	6	An integral type representing unsigned 8-bit integers with values between 0 and 255.
Char	4	An integral type representing unsigned 16-bit integers with values between 0 and 65535. The set of possible values for the Char type corresponds to the Unicode character set.
DateTime	16	A type representing a date and time value.
DBNull	2	A database null (column) value.
Decimal	15	A simple type representing values ranging from 1.0×10^{-28} to approximately 7.9×10^{28} with 28-29 significant digits.
Double	14	A floating point type representing values ranging from approximately 5.0×10^{-324} to 1.7×10^{308} with a precision of 15-16 digits.
Empty	0	A null reference.
Int16	7	An integral type representing signed 16-bit integers with values between -32768 and 32767.

Int32	9	An integral type representing signed 32-bit integers with values between -2147483648 and 2147483647.
Int64	11	An integral type representing signed 64-bit integers with values between -9223372036854775808 and 9223372036854775807.
Object	1	A general type representing any reference or value type not explicitly represented by another `TypeCode`.
SByte	5	An integral type representing signed 8-bit integers with values between -128 and 127.
Single	13	A floating point type representing values ranging from approximately 1.5×10^{-45} to 3.4×10^{38} with a precision of 7 digits.
String	18	A sealed class type representing Unicode character strings.
UInt16	8	An integral type representing unsigned 16-bit integers with values between 0 and 65535.
UInt32	10	An integral type representing unsigned 32-bit integers with values between 0 and 4294967295.
UInt64	12	An integral type representing unsigned 64-bit integers with values between 0 and 18446744073709551615.

Examples

The following code example demonstrates how the TypeCode enumeration can be used. In a decision block inside the WriteObjectInfo method, the TypeCode of an Object parameter is examined, and an appropriate message is written to the console.

C#

```
static void WriteObjectInfo(object testObject)
{
    TypeCode    typeCode = Type.GetTypeCode( testObject.GetType() );

    switch( typeCode )
    {
        case TypeCode.Boolean:
            Console.WriteLine("Boolean: {0}", testObject);
            break;

        case TypeCode.Double:
            Console.WriteLine("Double: {0}", testObject);
            break;

        default:
            Console.WriteLine("{0}: {1}", typeCode.ToString(), testObject);
```

```
                break;
        }
    }
```

Remarks

Call the GetTypeCode method on classes that implement the IConvertible interface to obtain the type code for an instance of that class.

Otherwise, call an object's GetType method to obtain its Type object, then call the `Type` object's GetTypeCode method to obtain the object's type code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypedReference Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Describes objects that contain both a managed pointer to a location and a runtime representation of the type that may be stored at that location.

C#

```
[System.CLSCompliant(false)]
public ref struct TypedReference
```

Inheritance Object → ValueType → TypedReference

Attributes CLSCompliantAttribute

Remarks

A typed reference is a type/value combination used for varargs and other support. `TypedReference` is a built-in value type that can be used for parameters and local variables.

Arrays of `TypedReference` objects cannot be created. For example, the following call is invalid:

C#

```
Assembly.Load("mscorlib.dll").GetType("System.TypedReference[]");
```

Methods

Expand table

Equals(Object)	Checks if this object is equal to the specified object.
GetHashCode()	Returns the hash code of this object.
GetTargetType(TypedReference)	Returns the type of the target of the specified `TypedReference`.
MakeTypedReference(Object, FieldInfo[])	Makes a `TypedReference` for a field identified by a specified object and list of field descriptions.
SetTypedReference(TypedReference, Object)	Converts the specified value to a `TypedReference`. This method is not supported.
TargetTypeToken(TypedReference)	Returns the internal metadata type handle for the specified `TypedReference`.
ToObject(TypedReference)	Converts the specified `TypedReference` to an `Object`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypedReference.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Checks if this object is equal to the specified object.

C#

```
public override bool Equals (object? o);
```

Parameters

o Object

The object with which to compare the current object.

Returns

Boolean

`true` if this object is equal to the specified object; otherwise, `false`.

Exceptions

NotSupportedException

This method is not implemented.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypedReference.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code of this object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code of this object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypedReference.GetTargetType(TypedReference) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the type of the target of the specified `TypedReference`.

```
C#
public static Type GetTargetType (TypedReference value);
```

Parameters

`value` TypedReference

The value whose target's type is to be returned.

Returns

Type

The type of the target of the specified `TypedReference`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypedReference.MakeTypedReference(Object, FieldInfo[]) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Makes a `TypedReference` for a field identified by a specified object and list of field descriptions.

C#

```
public static TypedReference MakeTypedReference (object target,
System.Reflection.FieldInfo[] flds);
```

Parameters

`target` Object

An object that contains the field described by the first element of `flds`.

`flds` FieldInfo[]

A list of field descriptions where each element describes a field that contains the field described by the succeeding element. Each described field must be a value type. The field descriptions must be `RuntimeFieldInfo` objects supplied by the type system.

Returns

TypedReference

A TypedReference for the field described by the last element of `flds`.

Exceptions

ArgumentNullException

`target` or `flds` is `null`.

-or-

An element of `flds` is `null`.

ArgumentException

The `flds` array has no elements.

-or-

An element of `flds` is not a `RuntimeFieldInfo` object.

-or-

The IsInitOnly or IsStatic property of an element of `flds` is `true`.

MissingMemberException

Parameter `target` does not contain the field described by the first element of `flds`, or an element of `flds` describes a field that is not contained in the field described by the succeeding element of `flds`.

-or-

The field described by an element of `flds` is not a value type.

Remarks

The MakeTypedReference method returns a typed reference to some terminal field, where the `target` parameter contains the field described by the first element of `flds`, the field described by the first element of `flds` contains the field described by the second element of `flds`, and so on until the terminal field is reached.

> **Note**
>
> Starting with the .NET Framework 2.0 Service Pack 1, this method can be used to access non-public members if the caller has been granted **ReflectionPermission** with the **ReflectionPermissionFlag.RestrictedMemberAccess** flag and if the grant set of the non-public members is restricted to the caller's grant set, or a subset thereof. (See **Security Considerations for Reflection**.)

To use this functionality, your application should target the .NET Framework 3.5 or later.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypedReference.SetTypedReference(TypedReference, Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the specified value to a `TypedReference`. This method is not supported.

C#

```
public static void SetTypedReference (TypedReference target, object? value);
```

Parameters

`target` TypedReference

The target of the conversion.

`value` Object

The value to be converted.

Exceptions

NotSupportedException

In all cases.

Remarks

This method always throws NotSupportedException.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypedReference.TargetTypeToken(TypedReference) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the internal metadata type handle for the specified `TypedReference`.

```
C#
public static RuntimeTypeHandle TargetTypeToken (TypedReference value);
```

Parameters

`value` TypedReference

The `TypedReference` for which the type handle is requested.

Returns

RuntimeTypeHandle

The internal metadata type handle for the specified `TypedReference`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypedReference.ToObject(TypedReference) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the specified `TypedReference` to an `Object`.

```
C#
public static object ToObject (TypedReference value);
```

Parameters

`value` TypedReference

The `TypedReference` to be converted.

Returns

Object

An Object converted from a `TypedReference`.

Remarks

This might be a boxing operation.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeInitializationException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown as a wrapper around the exception thrown by the class initializer. This class cannot be inherited.

C#

```
public sealed class TypeInitializationException : SystemException
```

Inheritance Object → Exception → SystemException → TypeInitializationException

Remarks

For more information about this API, see Supplemental API remarks for TypeInitializationException.

Constructors

Expand table

TypeInitializationException(String, Exception)	Initializes a new instance of the TypeInitializationException class with the default error message, the specified type name, and a reference to the inner exception that is the root cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception.

	(Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)
TypeName	Gets the fully qualified name of the type that fails to initialize.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the type name and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)

ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeInitializationException(String, Exception) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TypeInitializationException class with the default error message, the specified type name, and a reference to the inner exception that is the root cause of this exception.

```
C#

public TypeInitializationException (string? fullTypeName, Exception?
innerException);
```

Parameters

`fullTypeName` String

The fully qualified name of the type that fails to initialize.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of TypeInitializationException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The localized error message string.
TypeName	The name of the type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and throwing exceptions in .NET

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeInitializationException.TypeName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fully qualified name of the type that fails to initialize.

C#

```
public string TypeName { get; }
```

Property Value

String

The fully qualified name of the type that fails to initialize.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

TypeInitializationException.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the type name and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeLoadException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when type-loading failures occur.

C#

```
public class TypeLoadException : SystemException
```

Inheritance Object → Exception → SystemException → TypeLoadException

Derived System.DllNotFoundException
System.EntryPointNotFoundException
System.TypeAccessException

Remarks

TypeLoadException is thrown when the common language runtime cannot find the assembly, the type within the assembly, or cannot load the type. The Message property contains more detailed information that can help you identify the cause of the exception.

TypeLoadException uses the HRESULT COR_E_TYPELOAD, that has the value 0x80131522.

For a list of initial property values for an instance of TypeLoadException, see the TypeLoadException constructors.

The exact timing of when statically referenced types are loaded is unspecified. This exception may be thrown before the method that references the missing type starts executing.

Constructors

Expand table

TypeLoadException()	Initializes a new instance of the TypeLoadException class.
TypeLoadException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeLoadException class with serialized data.
TypeLoadException(String)	Initializes a new instance of the TypeLoadException class with a specified error message.
TypeLoadException(String, Exception)	Initializes a new instance of the TypeLoadException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets the error message for this exception.
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)
TypeName	Gets the fully qualified name of the type that causes the exception.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Sets the SerializationInfo object with the class name, method name, resource ID, and additional exception information.
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeLoadException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the TypeLoadException class.

Overloads

Expand table

TypeLoadException()	Initializes a new instance of the TypeLoadException class.
TypeLoadException(String)	Initializes a new instance of the TypeLoadException class with a specified error message.
TypeLoadException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeLoadException class with serialized data.
TypeLoadException(String, Exception)	Initializes a new instance of the TypeLoadException class with a specified error message and a reference to the inner exception that is the cause of this exception.

TypeLoadException()

Initializes a new instance of the TypeLoadException class.

C#

```
public TypeLoadException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "A failure has occurred while loading a type." This message takes into account the current system culture.

The following table shows the initial property values for an instance of TypeLoadException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TypeLoadException(String)

Initializes a new instance of the TypeLoadException class with a specified error message.

C#

```
public TypeLoadException (string? message);
```

Parameters

`message` String

The message that describes the error.

Examples

The following code example demonstrates the TypeLoadException(String) constructor. It contains a method that generates a TypeLoadException with a custom message, and displays the error message to the console.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      try {
         // Call a method that throws an exception.
         TypeLoadExceptionDemoClass.GenerateException();
      }
      catch (TypeLoadException e) {
         Console.WriteLine("TypeLoadException:\n   {0}", e.Message);
      }
   }
}

class TypeLoadExceptionDemoClass
{
   public static bool GenerateException()
   {
      // Throw a TypeLoadException with a custom defined message.
      throw new TypeLoadException("This is a custom TypeLoadException
error message.");
   }
}
// The example displays the following output:
//       TypeLoadException:
//          This is a custom TypeLoadException error message.
```

Remarks

The content of the `message` parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of TypeLoadException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TypeLoadException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the TypeLoadException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected TypeLoadException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is `null`.

Examples

The following example generates an exception, and serializes the exception data to a file, and then reconstitutes the exception. For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
using System.IO;

class GetObjectDataDemo
{
   public static void Main()
   {
      // Get a reference to the assembly mscorlib.dll, which is always
      // loaded. (System.String is defined in mscorlib.)
      Assembly mscorlib = typeof(string).Assembly;

      try
      {
         Console.WriteLine ("Attempting to load a type not present in the
assembly 'mscorlib'");
         // This loading of invalid type raises a TypeLoadException
         Type myType = mscorlib.GetType("System.NonExistentType", true);
      }
      catch (TypeLoadException)
      {
```

```
         // Serialize the exception to disk and reconstitute it.
         System.DateTime ErrorDatetime = DateTime.Now;
         Console.WriteLine("A TypeLoadException has been raised.");

         // Create MyTypeLoadException instance with current time.
         MyTypeLoadException myException = new
MyTypeLoadException(ErrorDatetime);
         IFormatter myFormatter = new SoapFormatter();
         Stream myFileStream = new FileStream("typeload.xml",
FileMode.Create, FileAccess.Write, FileShare.None);
         Console.WriteLine("Serializing the TypeLoadException with
DateTime as " + ErrorDatetime);

         // Serialize the MyTypeLoadException instance to a file.
         myFormatter.Serialize(myFileStream, myException);
         myFileStream.Close();

         Console.WriteLine("Deserializing the Exception.");
         myFileStream = new FileStream("typeload.xml", FileMode.Open,
FileAccess.Read, FileShare.None);

         // Deserialize and reconstitute the instance from file.
         myException = (MyTypeLoadException)
myFormatter.Deserialize(myFileStream);
         myFileStream.Close();
         Console.WriteLine("Deserialized exception has ErrorDateTime = "
+ myException.ErrorDateTime);
      }
   }
}

// This class overrides the GetObjectData method and initializes
// its data with current time.

[Serializable]
public class MyTypeLoadException : TypeLoadException
{
   private System.DateTime _errorDateTime = DateTime.Now;
   public DateTime ErrorDateTime { get { return _errorDateTime; }}

   public MyTypeLoadException(DateTime myDateTime)
   {
      _errorDateTime = myDateTime;
   }

   protected MyTypeLoadException(SerializationInfo sInfo,
StreamingContext sContext)
       : base(sInfo, sContext)
   {
      // Reconstitute the deserialized information into the instance.
      _errorDateTime = sInfo.GetDateTime("ErrorDate");
   }

   public override void GetObjectData(SerializationInfo sInfo,
StreamingContext sContext)
```

```
    {
        base.GetObjectData(sInfo, sContext);
        // Add a value to the Serialization information.
        sInfo.AddValue("ErrorDate", ErrorDateTime);
    }
}
```

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TypeLoadException(String, Exception)

Initializes a new instance of the TypeLoadException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public TypeLoadException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Examples

The following code example demonstrates the TypeLoadException(String, Exception) constructor. It contains a method that generates a TypeLoadException, catches that exception, and throws a new TypeLoadException with a custom message, including the original TypeLoadException as the inner exception.

C#

```
using System;
using System.Runtime.InteropServices;

public class TypeLoadException_Constructor3
{
   public static void Main()
   {
      Console.WriteLine("Calling a method in a non-existent DLL which
triggers a TypeLoadException.");
      try
      {
         TypeLoadExceptionDemoClass3.GenerateException();
      }
      catch (TypeLoadException e)
      {
         Console.WriteLine ("TypeLoadException: \n\tError Message = " +
e.Message);
         Console.WriteLine ("TypeLoadException: \n\tInnerException
Message = " + e.InnerException.Message );
      }
      catch (Exception e)
      {
         Console.WriteLine ("Exception: \n\tError Message = " +
e.Message);
      }
   }
}

class TypeLoadExceptionDemoClass3
{
   // A call to this method will raise a TypeLoadException.
   [DllImport("NonExistentDLL.DLL", EntryPoint="MethodNotExists")]
   public static extern void NonExistentMethod();

   public static void GenerateException()
```

```
    {
        try
        {
            NonExistentMethod();
        }
        catch (TypeLoadException e)
        {
            // Rethrow exception with the exception as inner exception
            throw new TypeLoadException("This exception was raised due to a
call to an invalid method.", e);
        }
    }
}
```

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of TypeLoadException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeLoadException.Message Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the error message for this exception.

C#

```
public override string Message { get; }
```

Property Value

String

The error message string.

Examples

The following example attempts to load a non-existent type from the mscorlib assembly. The resulting exception is caught, and the TypeName and Message values are displayed. For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;

public class TypeLoadException_TypeName
{
   public static void Main()
   {
      // Get a reference to the assembly mscorlib.dll, which is always
      // loaded. (System.String is defined in mscorlib.)
      Assembly mscorlib = typeof(string).Assembly;

      try
      {
```

```
            Console.WriteLine("Attempting to load a type that does not exist in
mscorlib.");
            // The boolean parameter causes an exception to be thrown if the
            // type is not found.
            Type myType = mscorlib.GetType("System.NonExistentType", true);
        }
        catch (TypeLoadException ex)
        {
            // Display the name of the type that was not found, and the
            // exception message.
            Console.WriteLine("TypeLoadException was caught. Type = '{0}'.",
                ex.TypeName);
            Console.WriteLine("Error Message = '{0}'", ex.Message);
        }
    }
}
/*
 This code example produces output similar to the following:

Attempting to load a type that does not exist in mscorlib.
TypeLoadException was caught. Type = 'System.NonExistentType'
Error Message = 'Could not load type System.NonExistentType from assembly
mscorl
ib, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089.'
 */
```

Remarks

This property overrides Message. The error message should be localized.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeLoadException.TypeName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the fully qualified name of the type that causes the exception.

C#

```
public string TypeName { get; }
```

Property Value

String

The fully qualified type name.

Examples

The following example attempts to load a non-existent type from the mscorlib assembly. The resulting exception is caught, and the TypeName and Message values are displayed. For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;

public class TypeLoadException_TypeName
{
   public static void Main()
   {
      // Get a reference to the assembly mscorlib.dll, which is always
      // loaded. (System.String is defined in mscorlib.)
      Assembly mscorlib = typeof(string).Assembly;

      try
      {
```

```
            Console.WriteLine("Attempting to load a type that does not exist in
mscorlib.");
            // The boolean parameter causes an exception to be thrown if the
            // type is not found.
            Type myType = mscorlib.GetType("System.NonExistentType", true);
        }
        catch (TypeLoadException ex)
        {
            // Display the name of the type that was not found, and the
            // exception message.
            Console.WriteLine("TypeLoadException was caught. Type = '{0}'.",
                ex.TypeName);
            Console.WriteLine("Error Message = '{0}'", ex.Message);
        }
    }
}
/*
 This code example produces output similar to the following:

Attempting to load a type that does not exist in mscorlib.
TypeLoadException was caught. Type = 'System.NonExistentType'
Error Message = 'Could not load type System.NonExistentType from assembly
mscorl
ib, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089.'
 */
```

Notes to Inheritors

When overriding TypeName in a derived class, be sure to call the base class's TypeName property.

This property is read-only.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeLoadException.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Sets the SerializationInfo object with the class name, method name, resource ID, and additional exception information.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public override void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `info` object is `null`.

Examples

The following example generates an exception, and serializes the exception data to a file, and then reconstitutes the exception. For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

C#

```
using System;
using System.Reflection;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
using System.IO;

class GetObjectDataDemo
{
   public static void Main()
   {
      // Get a reference to the assembly mscorlib.dll, which is always
      // loaded. (System.String is defined in mscorlib.)
      Assembly mscorlib = typeof(string).Assembly;

      try
      {
         Console.WriteLine ("Attempting to load a type not present in the
assembly 'mscorlib'");
         // This loading of invalid type raises a TypeLoadException
         Type myType = mscorlib.GetType("System.NonExistentType", true);
      }
      catch (TypeLoadException)
      {
         // Serialize the exception to disk and reconstitute it.
         System.DateTime ErrorDatetime = DateTime.Now;
         Console.WriteLine("A TypeLoadException has been raised.");

         // Create MyTypeLoadException instance with current time.
         MyTypeLoadException myException = new
MyTypeLoadException(ErrorDatetime);
         IFormatter myFormatter = new SoapFormatter();
         Stream myFileStream = new FileStream("typeload.xml",
FileMode.Create, FileAccess.Write, FileShare.None);
         Console.WriteLine("Serializing the TypeLoadException with DateTime
as " + ErrorDatetime);
```

```
         // Serialize the MyTypeLoadException instance to a file.
         myFormatter.Serialize(myFileStream, myException);
         myFileStream.Close();

         Console.WriteLine("Deserializing the Exception.");
         myFileStream = new FileStream("typeload.xml", FileMode.Open,
FileAccess.Read, FileShare.None);

         // Deserialize and reconstitute the instance from file.
         myException = (MyTypeLoadException)
myFormatter.Deserialize(myFileStream);
         myFileStream.Close();
         Console.WriteLine("Deserialized exception has ErrorDateTime = " +
myException.ErrorDateTime);
      }
   }
}

// This class overrides the GetObjectData method and initializes
// its data with current time.

[Serializable]
public class MyTypeLoadException : TypeLoadException
{
   private System.DateTime _errorDateTime = DateTime.Now;
   public DateTime ErrorDateTime { get { return _errorDateTime; }}

   public MyTypeLoadException(DateTime myDateTime)
   {
      _errorDateTime = myDateTime;
   }

   protected MyTypeLoadException(SerializationInfo sInfo, StreamingContext
sContext)
       : base(sInfo, sContext)
   {
      // Reconstitute the deserialized information into the instance.
      _errorDateTime = sInfo.GetDateTime("ErrorDate");
   }

   public override void GetObjectData(SerializationInfo sInfo,
StreamingContext sContext)
   {
      base.GetObjectData(sInfo, sContext);
      // Add a value to the Serialization information.
      sInfo.AddValue("ErrorDate", ErrorDateTime);
   }
}
```

Remarks

GetObjectData sets a SerializationInfo with all the exception object data targeted for serialization. During deserialization, the exception object is reconstituted from the SerializationInfo transmitted over the stream.

For more information, see XML and SOAP Serialization.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- SerializationInfo
- XML and SOAP Serialization

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

TypeUnloadedException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when there is an attempt to access an unloaded class.

C#

```
public class TypeUnloadedException : SystemException
```

Inheritance Object → Exception → SystemException → TypeUnloadedException

Remarks

TypeUnloadedException uses the HRESULT COR_E_TYPEUNLOADED, that has the value 0x80131013.

For a list of initial property values for an instance of TypeUnloadedException, see the TypeUnloadedException constructors.

Constructors

Expand table

TypeUnloadedException()	Initializes a new instance of the TypeUnloadedException class.
TypeUnloaded Exception(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeUnloadedException class with serialized data.
TypeUnloadedException(String)	Initializes a new instance of the TypeUnloadedException class with a specified error message.
TypeUnloadedException(String, Exception)	Initializes a new instance of the TypeUnloadedException class with a specified error message and a reference to the

	inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)

GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

TypeUnloadedException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes new instance of the TypeUnloadedException class.

Overloads

Expand table

TypeUnloadedException()	Initializes a new instance of the TypeUnloadedException class.
TypeUnloadedException(String)	Initializes a new instance of the TypeUnloadedException class with a specified error message.
TypeUnloadedException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the TypeUnloadedException class with serialized data.
TypeUnloadedException(String, Exception)	Initializes a new instance of the TypeUnloadedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

TypeUnloadedException()

Initializes a new instance of the TypeUnloadedException class.

C#

```
public TypeUnloadedException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Type has been unloaded." This message takes into account the current system culture.

The following table shows the initial property values for an instance of TypeUnloadedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TypeUnloadedException(String)

Initializes a new instance of the TypeUnloadedException class with a specified error message.

C#

```
public TypeUnloadedException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter should be understandable to the end user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of TypeUnloadedException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TypeUnloadedException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the TypeUnloadedException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected TypeUnloadedException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The object that holds the serialized object data.

`context` StreamingContext

The contextual information about the source or destination.

Attributes ObsoleteAttribute

Remarks

This constructor is called during deserialization to reconstitute the exception object transmitted over a stream. For more information, see XML and SOAP Serialization.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

TypeUnloadedException(String, Exception)

Initializes a new instance of the TypeUnloadedException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public TypeUnloadedException (string? message, Exception?
innerException);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`innerException` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception can include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of TypeUnloadedException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

See also

- Exception
- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents a 128-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UInt128 : IComparable<UInt128>, IEquatable<UInt128>,
IParsable<UInt128>, ISpanParsable<UInt128>, IUtf8SpanParsable<UInt128>,
System.Numerics.IAdditionOperators<UInt128,UInt128,UInt128>,
System.Numerics.IAdditiveIdentity<UInt128,UInt128>,
System.Numerics.IBinaryInteger<UInt128>,
System.Numerics.IBinaryNumber<UInt128>,
System.Numerics.IBitwiseOperators<UInt128,UInt128,UInt128>,
System.Numerics.IComparisonOperators<UInt128,UInt128,bool>,
System.Numerics.IDecrementOperators<UInt128>,
System.Numerics.IDivisionOperators<UInt128,UInt128,UInt128>,
System.Numerics.IEqualityOperators<UInt128,UInt128,bool>,
System.Numerics.IIncrementOperators<UInt128>,
System.Numerics.IMinMaxValue<UInt128>,
System.Numerics.IModulusOperators<UInt128,UInt128,UInt128>,
System.Numerics.IMultiplicativeIdentity<UInt128,UInt128>,
System.Numerics.IMultiplyOperators<UInt128,UInt128,UInt128>,
System.Numerics.INumber<UInt128>, System.Numerics.INumberBase<UInt128>,
System.Numerics.IShiftOperators<UInt128,int,UInt128>,
System.Numerics.ISubtractionOperators<UInt128,UInt128,UInt128>,
System.Numerics.IUnaryNegationOperators<UInt128,UInt128>,
System.Numerics.IUnaryPlusOperators<UInt128,UInt128>,
System.Numerics.IUnsignedNumber<UInt128>
```

Inheritance Object → ValueType → UInt128

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UInt128> , IComparable<TSelf> , IEquatable<UInt128> , IEquatable<TSelf> , IFormattable , IParsable<UInt128> , IParsable<TSelf> , ISpanFormattable , ISpanParsable<UInt128> , ISpanParsable<TSelf> , IAdditionOperators<UInt128,UInt128,UInt128> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UInt128,UInt128> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UInt128> , IBinaryNumber<UInt128> , IBinaryNumber<TSelf> , IBitwiseOperators<UInt128,UInt128,UInt128> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UInt128,UInt128,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UInt128> , IDecrementOperators<TSelf> , IDivisionOperators<UInt128,UInt128,UInt128> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UInt128,UInt128,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UInt128> , IIncrementOperators<TSelf> , IMinMaxValue<UInt128> , IModulusOperators<UInt128,UInt128,UInt128> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UInt128,UInt128> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UInt128,UInt128,UInt128> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UInt128> , INumber<TSelf> , INumberBase<UInt128> , INumberBase<TSelf> , IShiftOperators<UInt128,Int32,UInt128> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UInt128,UInt128,UInt128> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UInt128,UInt128> , IUnaryNegationOperators<TSelf,TSelf> ,

IUnaryPlusOperators<UInt128,UInt128> , IUnaryPlusOperators<TSelf,TSelf> , IUnsignedNumber<UInt128> , IUtf8SpanFormattable , IUtf8SpanParsable<UInt128> , IUtf8SpanParsable<TSelf>

Constructors

Expand table

UInt128(UInt64, UInt64)	Initializes a new instance of the UInt128 struct.

Properties

Expand table

MaxValue	Gets the maximum value of the current type.
MinValue	Gets the minimum value of the current type.
One	Gets the value `1` for the type.
Zero	Gets the value `0` for the type.

Methods

Expand table

Clamp(UInt128, UInt128, UInt128)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(UInt128)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall

	outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(UInt128, UInt128)	Computes the quotient and remainder of two values.
Equals(Object)	Determines whether the specified object is equal to the current object.
Equals(UInt128)	Indicates whether the current object is equal to another object of the same type.
GetHashCode()	Serves as the default hash function.
IsEvenInteger(UInt128)	Determines if a value represents an even integral number.
IsOddInteger(UInt128)	Determines if a value represents an odd integral number.
IsPow2(UInt128)	Determines if a value is a power of two.
LeadingZeroCount(UInt128)	Computes the number of leading zeros in a value.
Log2(UInt128)	Computes the log2 of a value.
Max(UInt128, UInt128)	Compares two values to compute which is greater.
Min(UInt128, UInt128)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
Parse(String)	Parses a string into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(String, NumberStyles)	Parses a string into a value.

Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.
PopCount(UInt128)	Computes the number of bits that are set in a value.
RotateLeft(UInt128, Int32)	Rotates a value left by a given amount.
RotateRight(UInt128, Int32)	Rotates a value right by a given amount.
Sign(UInt128)	Computes the sign of a value.
ToString()	Returns a string that represents the current object.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
TrailingZeroCount(UInt128)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt128)	Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt128)	Tries to parse a span of characters into a value.

TryParse(ReadOnlySpan<Char>, UInt128)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, UInt128)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, UInt128)	Tries to parse a string into a value.
TryParse(String, UInt128)	Tries to parse a string into a value.

Operators

Expand table

Addition(UInt128, UInt128)	Adds two values together to compute their sum.
BitwiseAnd(UInt128, UInt128)	Computes the bitwise-and of two values.
BitwiseOr(UInt128, UInt128)	Computes the bitwise-or of two values.
CheckedAddition(UInt128, UInt128)	Adds two values together to compute their sum.
CheckedDecrement(UInt128)	Decrements a value.
CheckedDivision(UInt128, UInt128)	Divides two values together to compute their quotient.
CheckedExplicit(Double)	
CheckedExplicit(Int16)	
CheckedExplicit(Int32)	
CheckedExplicit(Int64)	
CheckedExplicit(IntPtr)	
CheckedExplicit(SByte)	
CheckedExplicit(Single)	
CheckedExplicit(UInt128)	
CheckedIncrement(UInt128)	Increments a value.
CheckedMultiply(UInt128, UInt128)	Multiplies two values together to compute their product.
CheckedSubtraction(UInt128, UInt128)	Subtracts two values to compute their difference.

CheckedUnaryNegation(UInt128)	Computes the unary negation of a value.
Decrement(UInt128)	Decrements a value.
Division(UInt128, UInt128)	Divides two values together to compute their quotient.
Equality(UInt128, UInt128)	Compares two values to determine equality.
ExclusiveOr(UInt128, UInt128)	Computes the exclusive-or of two values.
Explicit(Decimal to UInt128)	Explicitly converts a Decimal value to a 128-bit unsigned integer.
Explicit(Double to UInt128)	Explicitly converts a Double value to a 128-bit unsigned integer.
Explicit(Int16 to UInt128)	Explicitly converts a Int16 value to a 128-bit unsigned integer.
Explicit(Int32 to UInt128)	Explicitly converts a Int32 value to a 128-bit unsigned integer.
Explicit(Int64 to UInt128)	Explicitly converts a Int64 value to a 128-bit unsigned integer.
Explicit(IntPtr to UInt128)	Explicitly converts a IntPtr value to a 128-bit unsigned integer.
Explicit(SByte to UInt128)	Explicitly converts a SByte value to a 128-bit unsigned integer.
Explicit(Single to UInt128)	Explicitly converts a Single value to a 128-bit unsigned integer.
Explicit(UInt128 to Byte)	Explicitly converts a 128-bit unsigned integer to a Byte value.
Explicit(UInt128 to Char)	Explicitly converts a 128-bit unsigned integer to a Char value.
Explicit(UInt128 to Decimal)	Explicitly converts a 128-bit unsigned integer to a Decimal value.
Explicit(UInt128 to Double)	Explicitly converts a 128-bit unsigned integer to a Double value.
Explicit(UInt128 to Half)	Explicitly converts a 128-bit unsigned integer to a Half value.
Explicit(UInt128 to Int128)	Explicitly converts a 128-bit unsigned integer to a Int128 value.

Explicit(UInt128 to Int16)	Explicitly converts a 128-bit unsigned integer to a Int16 value.
Explicit(UInt128 to Int32)	Explicitly converts a 128-bit unsigned integer to a Int32 value.
Explicit(UInt128 to Int64)	Explicitly converts a 128-bit unsigned integer to a Int64 value.
Explicit(UInt128 to IntPtr)	Explicitly converts a 128-bit unsigned integer to a IntPtr value.
Explicit(UInt128 to SByte)	Explicitly converts a 128-bit unsigned integer to a SByte value.
Explicit(UInt128 to Single)	Explicitly converts a 128-bit unsigned integer to a Single value.
Explicit(UInt128 to UInt16)	Explicitly converts a 128-bit unsigned integer to a UInt16 value.
Explicit(UInt128 to UInt32)	Explicitly converts a 128-bit unsigned integer to a UInt32 value.
Explicit(UInt128 to UInt64)	Explicitly converts a 128-bit unsigned integer to a UInt64 value.
Explicit(UInt128 to UIntPtr)	Explicitly converts a 128-bit unsigned integer to a UIntPtr value.
GreaterThan(UInt128, UInt128)	Compares two values to determine which is greater.
GreaterThanOrEqual(UInt128, UInt128)	Compares two values to determine which is greater or equal.
Implicit(Byte to UInt128)	Implicitly converts a Byte value to a 128-bit unsigned integer.
Implicit(Char to UInt128)	Implicitly converts a Char value to a 128-bit unsigned integer.
Implicit(UInt16 to UInt128)	Implicitly converts a UInt16 value to a 128-bit unsigned integer.
Implicit(UInt32 to UInt128)	Implicitly converts a UInt32 value to a 128-bit unsigned integer.
Implicit(UInt64 to UInt128)	Implicitly converts a UInt64 value to a 128-bit unsigned integer.
Implicit(UIntPtr to UInt128)	Implicitly converts a UIntPtr value to a 128-bit unsigned integer.

Increment(UInt128)	Increments a value.
Inequality(UInt128, UInt128)	Compares two values to determine inequality.
LeftShift(UInt128, Int32)	Shifts a value left by a given amount.
LessThan(UInt128, UInt128)	Compares two values to determine which is less.
LessThanOrEqual(UInt128, UInt128)	Compares two values to determine which is less or equal.
Modulus(UInt128, UInt128)	Divides two values together to compute their modulus or remainder.
Multiply(UInt128, UInt128)	Multiplies two values together to compute their product.
OnesComplement(UInt128)	Computes the ones-complement representation of a given value.
RightShift(UInt128, Int32)	Shifts a value right by a given amount.
Subtraction(UInt128, UInt128)	Subtracts two values to compute their difference.
UnaryNegation(UInt128)	Computes the unary negation of a value.
UnaryPlus(UInt128)	Computes the unary plus of a value.
UnsignedRightShift(UInt128, Int32)	Shifts a value right by a given amount.

Explicit Interface Implementations

Expand table

IAdditiveIdentity<UInt128,UInt128>.Additive Identity	Gets the additive identity of the current type.
IBinaryInteger<UInt128>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<UInt128>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<UInt128>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UInt128)	
IBinaryInteger<UInt128>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UInt128)	
IBinaryInteger<UInt128>.TryWriteBig Endian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.

IBinaryInteger<UInt128>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<UInt128>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IMultiplicativeIdentity<UInt128,UInt128>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
INumber<UInt128>.CopySign(UInt128, UInt128)	Copies the sign of a value to the sign of another value.
INumber<UInt128>.MaxNumber(UInt128, UInt128)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<UInt128>.MinNumber(UInt128, UInt128)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<UInt128>.Abs(UInt128)	Computes the absolute of a value.
INumberBase<UInt128>.IsCanonical(UInt128)	Determines if a value is in its canonical representation.
INumberBase<UInt128>.IsComplexNumber(UInt128)	Determines if a value represents a complex number.
INumberBase<UInt128>.IsFinite(UInt128)	Determines if a value is finite.
INumberBase<UInt128>.IsImaginaryNumber(UInt128)	Determines if a value represents a pure imaginary number.
INumberBase<UInt128>.IsInfinity(UInt128)	Determines if a value is infinite.
INumberBase<UInt128>.IsInteger(UInt128)	Determines if a value represents an integral number.
INumberBase<UInt128>.IsNaN(UInt128)	Determines if a value is NaN.
INumberBase<UInt128>.IsNegative(UInt128)	Determines if a value is negative.
INumberBase<UInt128>.IsNegativeInfinity(UInt128)	Determines if a value is negative infinity.
INumberBase<UInt128>.IsNormal(UInt128)	Determines if a value is normal.
INumberBase<UInt128>.IsPositive(UInt128)	Determines if a value is positive.
INumberBase<UInt128>.IsPositiveInfinity(UInt128)	Determines if a value is positive infinity.

INumberBase<UInt128>.IsRealNumber(UInt128)	Determines if a value represents a real number.
INumberBase<UInt128>.IsSubnormal(UInt128)	Determines if a value is subnormal.
INumberBase<UInt128>.IsZero(UInt128)	Determines if a value is zero.
INumberBase<UInt128>.MaxMagnitude(UInt128, UInt128)	Compares two values to compute which is greater.
INumberBase<UInt128>.MaxMagnitudeNumber(UInt128, UInt128)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt128>.MinMagnitude(UInt128, UInt128)	Compares two values to compute which is lesser.
INumberBase<UInt128>.MinMagnitudeNumber(UInt128, UInt128)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt128>.Radix	Gets the radix, or base, for the type.
INumberBase<UInt128>.TryConvertFromChecked<TOther>(TOther, UInt128)	
INumberBase<UInt128>.TryConvertFromSaturating<TOther>(TOther, UInt128)	
INumberBase<UInt128>.TryConvertFromTruncating<TOther>(TOther, UInt128)	
INumberBase<UInt128>.TryConvertToChecked<TOther>(UInt128, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<UInt128>.TryConvertToSaturating<TOther>(UInt128, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<UInt128>.TryConvertToTruncating<TOther>(UInt128, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128(UInt64, UInt64) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of the UInt128 struct.

C#

```
[System.CLSCompliant(false)]
public UInt128 (ulong upper, ulong lower);
```

Parameters

upper UInt64

The upper 64-bits of the 128-bit value.

lower UInt64

The lower 64-bits of the 128-bit value.

Attributes CLSCompliantAttribute

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
public static UInt128 MaxValue { get; }
```

Property Value

UInt128

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
public static UInt128 MinValue { get; }
```

Property Value

UInt128

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
public static UInt128 One { get; }
```

Property Value

UInt128

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
public static UInt128 Zero { get; }
```

Property Value

UInt128

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.Clamp(UInt128, UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static UInt128 Clamp (UInt128 value, UInt128 min, UInt128 max);
```

Parameters

`value` UInt128

The value to clamp.

`min` UInt128

The inclusive minimum to which `value` should clamp.

`max` UInt128

The inclusive maximum to which `value` should clamp.

Returns

UInt128

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(UInt128)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

CompareTo(Object)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CompareTo(UInt128)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (UInt128 value);
```

Parameters

`value` UInt128

An object to compare with this instance.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static UInt128 CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt128

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static UInt128 CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

UInt128

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static UInt128 CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

TOther

The type of `value`.

Parameters

value TOther

The value which is used to create the instance of `TSelf`.

Returns

UInt128

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.DivRem(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

C#

```
public static (UInt128 Quotient, UInt128 Remainder) DivRem (UInt128 left,
UInt128 right);
```

Parameters

left UInt128

The value which `right` divides.

right UInt128

The value which divides `left`.

Returns

ValueTuple<UInt128,UInt128>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(UInt128)	Indicates whether the current object is equal to another object of the same type.
Equals(Object)	Determines whether the specified object is equal to the current object.

Equals(UInt128)

Indicates whether the current object is equal to another object of the same type.

C#

```
public bool Equals (UInt128 other);
```

Parameters

other UInt128

An object to compare with this object.

Returns

Boolean

`true` if the current object is equal to the `other` parameter; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Equals(Object)

Determines whether the specified object is equal to the current object.

```
C#
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with the current object.

Returns

Boolean

`true` if the specified object is equal to the current object; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt128.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Serves as the default hash function.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A hash code for the current object.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.IsEvenInteger(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.IsOddInteger(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.IsPow2(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

```
C#
public static bool IsPow2 (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.LeadingZeroCount(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

```
C#
public static UInt128 LeadingZeroCount (UInt128 value);
```

Parameters

`value` UInt128

The value whose leading zeroes are to be counted.

Returns

UInt128

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Log2(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static UInt128 Log2 (UInt128 value);
```

Parameters

`value` UInt128

The value whose log2 is to be computed.

Returns

UInt128

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Max(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static UInt128 Max (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Min(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static UInt128 Min (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String)	Parses a string into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, NumberStyles)	Parses a string into a value.
Parse(String, IFormatProvider)	Parses a string into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Parses a span of characters into a value.
Parse(String, NumberStyles, IFormatProvider)	Parses a string into a value.

Parse(String)

Parses a string into a value.

C#

```
public static UInt128 Parse (string s);
```

Parameters

s String

A string containing a number to parse.

Returns

UInt128

The result of parsing `s`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static UInt128 Parse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt128

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static UInt128 Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt128

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles)

Parses a string into a value.

```
C#

public static UInt128 Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

Returns

UInt128

The result of parsing `s`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, IFormatProvider)

Parses a string into a value.

```
C#

public static UInt128 Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt128

The result of parsing `s`.

Implements

Parse(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static UInt128 Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt128

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static UInt128 Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt128

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles, IFormatProvider)

Parses a string into a value.

C#

```
public static UInt128 Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

The string to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt128

The result of parsing `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.PopCount(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static UInt128 PopCount (UInt128 value);
```

Parameters

`value` UInt128

The value whose set bits are to be counted.

Returns

UInt128

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.RotateLeft(UInt128, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static UInt128 RotateLeft (UInt128 value, int rotateAmount);
```

Parameters

`value` UInt128

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

UInt128

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.RotateRight(UInt128, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static UInt128 RotateRight (UInt128 value, int rotateAmount);
```

Parameters

`value` UInt128

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

UInt128

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Sign(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (UInt128 value);
```

Parameters

`value` UInt128

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
ToString()	Returns a string that represents the current object.

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

C#

```
public string ToString (string? format);
```

Parameters

`format` String

The format to use, or a `null` reference to use the default format defined for the type of the IFormattable implementation.

Returns

String

The string representation of the value of this instance as specified by `format`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString(String, IFormatProvider)

Formats the value of the current instance using the specified format.

```
C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

The format to use, or a `null` reference to use the default format defined for the type of the IFormattable implementation.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

ToString()

Returns a string that represents the current object.

```
C#
```

```
public override string ToString ();
```

Returns

String

A string that represents the current object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.TrailingZeroCount(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static UInt128 TrailingZeroCount (UInt128 value);
```

Parameters

`value` UInt128

The value whose trailing zeroes are to be counted.

Returns

UInt128

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt128.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(ReadOnlySpan<Char>, UInt128)	Tries to parse a span of characters into a value.
TryParse(String, UInt128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt128)	Tries to parse a string into a value.
TryParse(String, IFormatProvider, UInt128)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt128)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt128)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt128)	Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit unsigned integer equivalent.
TryParse(String, NumberStyles, IFormatProvider, UInt128)	Tries to parse a string into a value.

TryParse(ReadOnlySpan<Char>, UInt128)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, out UInt128 result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

result UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, UInt128)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, out UInt128 result);
```

Parameters

s String

The string to parse.

result UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt128)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out UInt128 result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt128

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt128)

Tries to parse a string into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out UInt128 result);
```

Parameters

s ReadOnlySpan<Char>

The string to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, UInt128)

Tries to parse a string into a value.

C#

```
public static bool TryParse (string? s, IFormatProvider? provider, out
UInt128 result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt128)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
UInt128 result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt128

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt128)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
UInt128 result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, UInt128)

Tries to convert a UTF-8 character span containing the string representation of a number to its 128-bit unsigned integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out UInt128
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` UInt128

When this method returns, contains the 128-bit unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, NumberStyles, IFormatProvider, UInt128)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out UInt128 result);
```

Parameters

`s` String

The string to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `s`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt128

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Addition(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#
public static UInt128 operator + (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to which `right` is added.

`right` UInt128

The value which is added to `left`.

Returns

UInt128

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.BitwiseAnd(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#
public static UInt128 operator & (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to `and` with `right`.

`right` UInt128

The value to `and` with `left`.

Returns

UInt128

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.BitwiseOr(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#
public static UInt128 operator | (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to `or` with `right`.

`right` UInt128

The value to `or` with `left`.

Returns

UInt128

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedAddition(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#
public static UInt128 op_CheckedAddition (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to which `right` is added.

`right` UInt128

The value which is added to `left`.

Returns

UInt128

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedDecrement(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
public static UInt128 op_CheckedDecrement (UInt128 value);
```

Parameters

`value` UInt128

The value to decrement.

Returns

UInt128

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedDivision(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their quotient.

```
C#
public static UInt128 op_CheckedDivision (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value which `right` divides.

`right` UInt128

The value which divides `left`.

Returns

UInt128

The quotient of `left` divided-by `right`.

Implements

CheckedDivision(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedExplicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CheckedExplicit(UInt128)
CheckedExplicit(Single)
CheckedExplicit(SByte)
CheckedExplicit(IntPtr)
CheckedExplicit(Int16)
CheckedExplicit(Int32)
CheckedExplicit(Double)
CheckedExplicit(Int64)

CheckedExplicit(UInt128)

C#

```
public static byte op_CheckedExplicit (UInt128 value);
```

Parameters

`value` UInt128

Returns

Byte

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Single)

C#

```
public static UInt128 op_CheckedExplicit (float value);
```

Parameters

`value` Single

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(SByte)

> Important
>
> This API is not CLS-compliant.

C#

```
[System.CLSCompliant(false)]
public static UInt128 op_CheckedExplicit (sbyte value);
```

Parameters

`value` SByte

Returns

UInt128

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(IntPtr)

C#

```
public static UInt128 op_CheckedExplicit (IntPtr value);
```

Parameters

`value` IntPtr

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Int16)

```
C#
public static UInt128 op_CheckedExplicit (short value);
```

Parameters

`value` Int16

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Int32)

```
C#
public static UInt128 op_CheckedExplicit (int value);
```

Parameters

`value` Int32

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Double)

```
C#

public static UInt128 op_CheckedExplicit (double value);
```

Parameters

`value` Double

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

CheckedExplicit(Int64)

```
C#

public static UInt128 op_CheckedExplicit (long value);
```

Parameters

`value` Int64

Returns

UInt128

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedIncrement(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
public static UInt128 op_CheckedIncrement (UInt128 value);
```

Parameters

`value` UInt128

The value to increment.

Returns

UInt128

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedMultiply(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

C#

```
public static UInt128 op_CheckedMultiply (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value which `right` multiplies.

`right` UInt128

The value which multiplies `left`.

Returns

UInt128

The product of `left` multiplied-by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedSubtraction(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#
public static UInt128 op_CheckedSubtraction (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value from which `right` is subtracted.

`right` UInt128

The value which is subtracted from `left`.

Returns

UInt128

The difference of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.CheckedUnaryNegation(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
public static UInt128 op_CheckedUnaryNegation (UInt128 value);
```

Parameters

`value` UInt128

The value for which to compute the unary negation.

Returns

UInt128

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Decrement(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
public static UInt128 operator -- (UInt128 value);
```

Parameters

`value` UInt128

The value to decrement.

Returns

UInt128

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Division(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their quotient.

```
C#
public static UInt128 operator / (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value which `right` divides.

`right` UInt128

The value which divides `left`.

Returns

UInt128

The quotient of `left` divided-by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Equality(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#
public static bool operator == (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.ExclusiveOr(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#
public static UInt128 operator ^ (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to xor with `right`.

`right` UInt128

The value to xor with `left`.

Returns

UInt128

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Explicit(UInt128 to UIntPtr)	Explicitly converts a 128-bit unsigned integer to a UIntPtr value.
Explicit(UInt128 to UInt64)	Explicitly converts a 128-bit unsigned integer to a UInt64 value.
Explicit(UInt128 to UInt32)	Explicitly converts a 128-bit unsigned integer to a UInt32 value.
Explicit(UInt128 to UInt16)	Explicitly converts a 128-bit unsigned integer to a UInt16 value.
Explicit(UInt128 to Single)	Explicitly converts a 128-bit unsigned integer to a Single value.
Explicit(UInt128 to SByte)	Explicitly converts a 128-bit unsigned integer to a SByte value.
Explicit(UInt128 to IntPtr)	Explicitly converts a 128-bit unsigned integer to a IntPtr value.
Explicit(UInt128 to Int64)	Explicitly converts a 128-bit unsigned integer to a Int64 value.
Explicit(UInt128 to Int32)	Explicitly converts a 128-bit unsigned integer to a Int32 value.
Explicit(UInt128 to Int128)	Explicitly converts a 128-bit unsigned integer to a Int128 value.
Explicit(UInt128 to Half)	Explicitly converts a 128-bit unsigned integer to a Half value.
Explicit(UInt128 to Int16)	Explicitly converts a 128-bit unsigned integer to a Int16 value.
Explicit(UInt128 to Decimal)	Explicitly converts a 128-bit unsigned integer to a Decimal value.
Explicit(Decimal to UInt128)	Explicitly converts a Decimal value to a 128-bit unsigned integer.
Explicit(UInt128 to Double)	Explicitly converts a 128-bit unsigned integer to a Double value.
Explicit(Int16 to UInt128)	Explicitly converts a Int16 value to a 128-bit unsigned integer.
Explicit(Int32 to UInt128)	Explicitly converts a Int32 value to a 128-bit unsigned integer.
Explicit(Int64 to UInt128)	Explicitly converts a Int64 value to a 128-bit unsigned integer.

Explicit(Double to UInt128)	Explicitly converts a Double value to a 128-bit unsigned integer.
Explicit(SByte to UInt128)	Explicitly converts a SByte value to a 128-bit unsigned integer.
Explicit(Single to UInt128)	Explicitly converts a Single value to a 128-bit unsigned integer.
Explicit(UInt128 to Byte)	Explicitly converts a 128-bit unsigned integer to a Byte value.
Explicit(UInt128 to Char)	Explicitly converts a 128-bit unsigned integer to a Char value.
Explicit(IntPtr to UInt128)	Explicitly converts a IntPtr value to a 128-bit unsigned integer.

Explicit(UInt128 to UIntPtr)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a UIntPtr value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator UIntPtr (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

UIntPtr

`value` converted to a UIntPtr.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to UInt64)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a UInt64 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator ulong (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

UInt64

`value` converted to a UInt64.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to UInt32)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a UInt32 value.

C#

```
[System.CLSCompliant(false)]
public static explicit operator uint (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

UInt32

`value` converted to a UInt32.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to UInt16)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a UInt16 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator ushort (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

UInt16

`value` converted to a UInt16.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Single)

Explicitly converts a 128-bit unsigned integer to a Single value.

```
C#

public static explicit operator float (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Single

`value` converted to a Single.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to SByte)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a SByte value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator sbyte (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

SByte

`value` converted to a SByte.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to IntPtr)

Explicitly converts a 128-bit unsigned integer to a IntPtr value.

```
C#
public static explicit operator IntPtr (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

IntPtr

`value` converted to a IntPtr.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Int64)

Explicitly converts a 128-bit unsigned integer to a Int64 value.

```
C#
public static explicit operator long (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Int64

`value` converted to a Int64.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Int32)

Explicitly converts a 128-bit unsigned integer to a Int32 value.

```
C#
public static explicit operator int (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Int32

`value` converted to a Int32.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Int128)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a 128-bit unsigned integer to a Int128 value.

```
C#

[System.CLSCompliant(false)]
public static explicit operator Int128 (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Int128

`value` converted to a Int128.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Half)

Explicitly converts a 128-bit unsigned integer to a Half value.

```
C#
public static explicit operator Half (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Half

`value` converted to a Half.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Int16)

Explicitly converts a 128-bit unsigned integer to a Int16 value.

```
C#
public static explicit operator short (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Int16

`value` converted to a Int16.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Decimal)

Explicitly converts a 128-bit unsigned integer to a Decimal value.

```
C#

public static explicit operator decimal (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Decimal

`value` converted to a Decimal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Decimal to UInt128)

Explicitly converts a Decimal value to a 128-bit unsigned integer.

```
C#
public static explicit operator UInt128 (decimal value);
```

Parameters

`value` Decimal

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Double)

Explicitly converts a 128-bit unsigned integer to a Double value.

```
C#
public static explicit operator double (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Double

`value` converted to a Double.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int16 to UInt128)

Explicitly converts a Int16 value to a 128-bit unsigned integer.

C#

```
public static explicit operator UInt128 (short value);
```

Parameters

`value` Int16

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int32 to UInt128)

Explicitly converts a Int32 value to a 128-bit unsigned integer.

C#

```
public static explicit operator UInt128 (int value);
```

Parameters

`value` Int32

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Int64 to UInt128)

Explicitly converts a Int64 value to a 128-bit unsigned integer.

C#

```
public static explicit operator UInt128 (long value);
```

Parameters

`value` Int64

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Double to UInt128)

Explicitly converts a Double value to a 128-bit unsigned integer.

C#

```
public static explicit operator UInt128 (double value);
```

Parameters

`value` Double

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(SByte to UInt128)

> Important
>
> This API is not CLS-compliant.

Explicitly converts a SByte value to a 128-bit unsigned integer.

```
C#

[System.CLSCompliant(false)]
public static explicit operator UInt128 (sbyte value);
```

Parameters

`value` SByte

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(Single to UInt128)

Explicitly converts a Single value to a 128-bit unsigned integer.

```
C#

public static explicit operator UInt128 (float value);
```

Parameters

`value` Single

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Byte)

Explicitly converts a 128-bit unsigned integer to a Byte value.

```
C#
public static explicit operator byte (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Byte

`value` converted to a Byte.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(UInt128 to Char)

Explicitly converts a 128-bit unsigned integer to a Char value.

```
C#
public static explicit operator char (UInt128 value);
```

Parameters

`value` UInt128

The value to convert.

Returns

Char

`value` converted to a Char.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Explicit(IntPtr to UInt128)

Explicitly converts a IntPtr value to a 128-bit unsigned integer.

```
C#
public static explicit operator UInt128 (IntPtr value);
```

Parameters

`value` IntPtr

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.GreaterThan(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#
public static bool operator > (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.GreaterThanOrEqual(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#
public static bool operator >= (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Implicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Implicit(Byte to UInt128)	Implicitly converts a Byte value to a 128-bit unsigned integer.
Implicit(Char to UInt128)	Implicitly converts a Char value to a 128-bit unsigned integer.
Implicit(UInt16 to UInt128)	Implicitly converts a UInt16 value to a 128-bit unsigned integer.
Implicit(UInt32 to UInt128)	Implicitly converts a UInt32 value to a 128-bit unsigned integer.
Implicit(UInt64 to UInt128)	Implicitly converts a UInt64 value to a 128-bit unsigned integer.
Implicit(UIntPtr to UInt128)	Implicitly converts a UIntPtr value to a 128-bit unsigned integer.

Implicit(Byte to UInt128)

Implicitly converts a Byte value to a 128-bit unsigned integer.

C#

```
public static implicit operator UInt128 (byte value);
```

Parameters

`value` Byte

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(Char to UInt128)

Implicitly converts a Char value to a 128-bit unsigned integer.

C#

```
public static implicit operator UInt128 (char value);
```

Parameters

`value` Char

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt16 to UInt128)

Important

This API is not CLS-compliant.

Implicitly converts a UInt16 value to a 128-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator UInt128 (ushort value);
```

Parameters

`value` UInt16

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt32 to UInt128)

Important

This API is not CLS-compliant.

Implicitly converts a UInt32 value to a 128-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator UInt128 (uint value);
```

Parameters

`value` UInt32

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UInt64 to UInt128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UInt64 value to a 128-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator UInt128 (ulong value);
```

Parameters

`value` UInt64

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Implicit(UIntPtr to UInt128)

> Important
>
> This API is not CLS-compliant.

Implicitly converts a UIntPtr value to a 128-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public static implicit operator UInt128 (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to convert.

Returns

UInt128

`value` converted to a 128-bit unsigned integer.

Attributes CLSCompliantAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.Increment(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
public static UInt128 operator ++ (UInt128 value);
```

Parameters

`value` UInt128

The value to increment.

Returns

UInt128

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Inequality(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#
public static bool operator != (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.LeftShift(UInt128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

C#

```
public static UInt128 operator << (UInt128 value, int shiftAmount);
```

Parameters

`value` UInt128

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

UInt128

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.LessThan(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#
public static bool operator < (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.LessThanOrEqual(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#
public static bool operator <= (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value to compare with `right`.

`right` UInt128

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Modulus(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#
public static UInt128 operator % (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value which `right` divides.

`right` UInt128

The value which divides `left`.

Returns

UInt128

The modulus or remainder of `left` divided-by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Multiply(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#
public static UInt128 operator * (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value which `right` multiplies.

`right` UInt128

The value which multiplies `left`.

Returns

UInt128

The product of `left` multiplied-by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.OnesComplement(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

C#

```
public static UInt128 operator ~ (UInt128 value);
```

Parameters

`value` UInt128

The value for which to compute the ones-complement.

Returns

UInt128

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.RightShift(UInt128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#
public static UInt128 operator >> (UInt128 value, int shiftAmount);
```

Parameters

`value` UInt128

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt128

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Remarks

This operation is meant to perform a signed (otherwise known as an arithmetic) right shift on signed types.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.Subtraction(UInt128, UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#
public static UInt128 operator - (UInt128 left, UInt128 right);
```

Parameters

`left` UInt128

The value from which `right` is subtracted.

`right` UInt128

The value which is subtracted from `left`.

Returns

UInt128

The difference of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.UnaryNegation(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
public static UInt128 operator - (UInt128 value);
```

Parameters

`value` UInt128

The value for which to compute the unary negation.

Returns

UInt128

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.UnaryPlus(UInt128) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
public static UInt128 operator + (UInt128 value);
```

Parameters

`value` UInt128

The value for which to compute the unary plus.

Returns

UInt128

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.UnsignedRightShift(UInt128, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
public static UInt128 op_UnsignedRightShift (UInt128 value, int
shiftAmount);
```

Parameters

`value` UInt128

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt128

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Remarks

This operation is meant to perform n unsigned (otherwise known as a logical) right shift on all types.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.IAdditiveIdentity<UInt128,UInt128>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static UInt128
System.Numerics.IAdditiveIdentity<System.UInt128,System.UInt128>.AdditiveIde
ntity { get; }
```

Property Value

UInt128

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt128.IBinaryInteger<UInt128>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<UInt128>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt128.IBinaryInteger<UInt128>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<UInt128>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt128.IBinaryInteger<UInt128>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<UInt128>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out UInt128 value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt128.IBinaryInteger<UInt128>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<UInt128>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out UInt128 value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt128.IBinaryInteger<UInt128>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<UInt128>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.IBinaryInteger<UInt128>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<UInt128>.TryWriteLittleEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.IBinaryNumber<UInt128>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static UInt128 System.Numerics.IBinaryNumber<System.UInt128>.AllBitsSet {
get; }
```

Property Value

UInt128

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt128.IMultiplicativeIdentity<UInt128,UInt128>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static UInt128
System.Numerics.IMultiplicativeIdentity<System.UInt128,System.UInt128>.MultiplicativeIdentity { get; }
```

Property Value

UInt128

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt128.INumber<UInt128>.CopySign(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static UInt128 INumber<UInt128>.CopySign (UInt128 value, UInt128 sign);
```

Parameters

`value` UInt128

The value whose magnitude is used in the result.

`sign` UInt128

The value whose sign is used in the result.

Returns

UInt128

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumber<UInt128>.MaxNumber(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

C#

```
static UInt128 INumber<UInt128>.MaxNumber (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumber<UInt128>.MinNumber(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

C#

```
static UInt128 INumber<UInt128>.MinNumber (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.Abs(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static UInt128 INumberBase<UInt128>.Abs (UInt128 value);
```

Parameters

`value` UInt128

The value for which to get its absolute.

Returns

UInt128

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsCanonical(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

C#

```
static bool INumberBase<UInt128>.IsCanonical (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsComplexNumber(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

C#

```
static bool INumberBase<UInt128>.IsComplexNumber (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsFinite(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<UInt128>.IsFinite (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsImaginaryNumber(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<UInt128>.IsImaginaryNumber (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsInfinity(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

C#

```
static bool INumberBase<UInt128>.IsInfinity (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsInteger(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

C#

```
static bool INumberBase<UInt128>.IsInteger (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsNaN(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<UInt128>.IsNaN (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsNegative(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<UInt128>.IsNegative (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsNegativeInfinity(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<UInt128>.IsNegativeInfinity (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsNormal(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<UInt128>.IsNormal (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsPositive(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<UInt128>.IsPositive (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsPositiveInfinity(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<UInt128>.IsPositiveInfinity (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsRealNumber(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

C#

```
static bool INumberBase<UInt128>.IsRealNumber (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsSubnormal(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<UInt128>.IsSubnormal (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.IsZero(UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<UInt128>.IsZero (UInt128 value);
```

Parameters

`value` UInt128

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.MaxMagnitude(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static UInt128 INumberBase<UInt128>.MaxMagnitude (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.MaxMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static UInt128 INumberBase<UInt128>.MaxMagnitudeNumber (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.MinMagnitude(UInt128, UInt128) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
static UInt128 INumberBase<UInt128>.MinMagnitude (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.MinMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

C#

```
static UInt128 INumberBase<UInt128>.MinMagnitudeNumber (UInt128 x, UInt128 y);
```

Parameters

x UInt128

The value to compare with `y`.

y UInt128

The value to compare with `x`.

Returns

UInt128

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.UInt128>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UInt128>.TryConvertFromChecked<TOther> (TOther
value, out UInt128 result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UInt128>.TryConvertFromSaturating<TOther> (TOther
value, out UInt128 result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UInt128>.TryConvertFromTruncating<TOther> (TOther
value, out UInt128 result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt128

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<UInt128>.TryConvertToChecked<TOther> (UInt128 value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<UInt128>.TryConvertToSaturating<TOther> (UInt128
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt128.INumberBase<UInt128>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<UInt128>.TryConvertToTruncating<TOther> (UInt128
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt128

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents a 16-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UInt16 : IComparable<ushort>, IConvertible,
IEquatable<ushort>, IParsable<ushort>, ISpanParsable<ushort>,
IUtf8SpanParsable<ushort>,
System.Numerics.IAdditionOperators<ushort,ushort,ushort>,
System.Numerics.IAdditiveIdentity<ushort,ushort>,
System.Numerics.IBinaryInteger<ushort>,
System.Numerics.IBinaryNumber<ushort>,
System.Numerics.IBitwiseOperators<ushort,ushort,ushort>,
System.Numerics.IComparisonOperators<ushort,ushort,bool>,
System.Numerics.IDecrementOperators<ushort>,
System.Numerics.IDivisionOperators<ushort,ushort,ushort>,
System.Numerics.IEqualityOperators<ushort,ushort,bool>,
System.Numerics.IIncrementOperators<ushort>,
System.Numerics.IMinMaxValue<ushort>,
System.Numerics.IModulusOperators<ushort,ushort,ushort>,
System.Numerics.IMultiplicativeIdentity<ushort,ushort>,
System.Numerics.IMultiplyOperators<ushort,ushort,ushort>,
System.Numerics.INumber<ushort>, System.Numerics.INumberBase<ushort>,
System.Numerics.IShiftOperators<ushort,int,ushort>,
System.Numerics.ISubtractionOperators<ushort,ushort,ushort>,
System.Numerics.IUnaryNegationOperators<ushort,ushort>,
System.Numerics.IUnaryPlusOperators<ushort,ushort>,
System.Numerics.IUnsignedNumber<ushort>
```

Inheritance Object → ValueType → UInt16

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UInt16> , IConvertible , IEquatable<UInt16> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<UInt16> , IParsable<TSelf> , ISpanParsable<UInt16> , ISpanParsable<TSelf> , IAdditionOperators<UInt16,UInt16,UInt16> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UInt16,UInt16> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UInt16> , IBinaryNumber<UInt16> , IBinaryNumber<TSelf> , IBitwiseOperators<UInt16,UInt16,UInt16> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UInt16,UInt16,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UInt16> , IDecrementOperators<TSelf> , IDivisionOperators<UInt16,UInt16,UInt16> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UInt16,UInt16,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UInt16> , IIncrementOperators<TSelf> , IMinMaxValue<UInt16> , IModulusOperators<UInt16,UInt16,UInt16> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UInt16,UInt16> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UInt16,UInt16,UInt16> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UInt16> , INumber<TSelf> , INumberBase<UInt16> , INumberBase<TSelf> , IShiftOperators<UInt16,Int32,UInt16> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UInt16,UInt16,UInt16> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UInt16,UInt16> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<UInt16,UInt16> , IUnaryPlusOperators<TSelf,TSelf> ,

IUnsignedNumber<UInt16> , IUtf8SpanFormattable ,
IUtf8SpanParsable<UInt16> , IUtf8SpanParsable<TSelf>

Remarks

The UInt16 value type represents unsigned integers with values ranging from 0 to 65535.

> **Important**
>
> The **UInt16** type is not CLS-compliant. The CLS-compliant alternative type is **Int32**. **Int16** can be used instead to replace a **UInt16** value that ranges from zero to **Int16.MaxValue**. For more information about CLS compliance, see **Language Independence and Language-Independent Components**.

The UInt16 structure provides methods to compare instances of this type, convert the value of an instance to its string representation, and convert the string representation of a number to an instance of this type.

For information about how format specification codes control the string representation of value types, see Formatting Types.

Fields

Expand table

MaxValue	Represents the largest possible value of UInt16. This field is constant.
MinValue	Represents the smallest possible value of UInt16. This field is constant.

Methods

Expand table

Clamp(UInt16, UInt16, UInt16)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(UInt16)	Compares this instance to a specified 16-bit unsigned integer and returns an indication of their relative values.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(UInt16, UInt16)	Computes the quotient and remainder of two values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt16)	Returns a value indicating whether this instance is equal to a specified UInt16 value.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type UInt16.
IsEvenInteger(UInt16)	Determines if a value represents an even integral number.
IsOddInteger(UInt16)	Determines if a value represents an odd integral number.
IsPow2(UInt16)	Determines if a value is a power of two.
LeadingZeroCount(UInt16)	Computes the number of leading zeros in a value.
Log2(UInt16)	Computes the log2 of a value.
Max(UInt16, UInt16)	Compares two values to compute which is greater.
Min(UInt16, UInt16)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned

	integer equivalent.
Parse(String)	Converts the string representation of a number to its 16-bit unsigned integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 16-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 16-bit unsigned integer equivalent. This method is not CLS-compliant. The CLS-compliant alternative is Parse(String, NumberStyles).
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent.
PopCount(UInt16)	Computes the number of bits that are set in a value.
RotateLeft(UInt16, Int32)	Rotates a value left by a given amount.
RotateRight(UInt16, Int32)	Rotates a value right by a given amount.
Sign(UInt16)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(UInt16)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current unsigned short number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>,	Tries to parse a span of UTF-8 characters into a value.

IFormatProvider, UInt16)	
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt16)	Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt16)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt16)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, UInt16)	Tries to convert the span representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, UInt16)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, UInt16)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt16)	Tries to convert the string representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<UInt16,UInt16,UInt16>.Addition(UInt16, UInt16)	Adds two values together to compute their sum.
IAdditionOperators<UInt16,UInt16,UInt16>.CheckedAddition(UInt16, UInt16)	Adds two values together to compute their sum.
IAdditiveIdentity<UInt16,UInt16>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<UInt16>.GetByteCount()	Gets the number of bytes that will be written as part of

	TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<UInt16>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<UInt16>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UInt16)	
IBinaryInteger<UInt16>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UInt16)	
IBinaryInteger<UInt16>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<UInt16>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<UInt16>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<UInt16,UInt16,UInt16>.BitwiseAnd(UInt16, UInt16)	Computes the bitwise-and of two values.
IBitwiseOperators<UInt16,UInt16,UInt16>.BitwiseOr(UInt16, UInt16)	Computes the bitwise-or of two values.
IBitwiseOperators<UInt16,UInt16,UInt16>.ExclusiveOr(UInt16, UInt16)	Computes the exclusive-or of two values.
IBitwiseOperators<UInt16,UInt16,UInt16>.OnesComplement(UInt16)	Computes the ones-complement representation of a given value.
IComparisonOperators<UInt16,UInt16,Boolean>.GreaterThan(UInt16, UInt16)	Compares two values to determine which is greater.
IComparisonOperators<UInt16,UInt16,Boolean>.GreaterThanOrEqual(UInt16, UInt16)	Compares two values to determine which is greater or equal.
IComparisonOperators<UInt16,UInt16,Boolean>.LessThan(UInt16, UInt16)	Compares two values to determine which is less.
IComparisonOperators<UInt16,UInt16,Boolean>.LessThanOrEqual(UInt16, UInt16)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member,

	see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<UInt16>.CheckedDecrement(UInt16)	Decrements a value.
IDecrementOperators<UInt16>.Decrement(UInt16)	Decrements a value.
IDivisionOperators<UInt16,UInt16,UInt16>.Division(UInt16, UInt16)	Divides one value by another to compute their quotient.
IEqualityOperators<UInt16,UInt16,Boolean>.Equality(UInt16, UInt16)	Compares two values to determine equality.
IEqualityOperators<UInt16,UInt16,Boolean>.Inequality(UInt16, UInt16)	Compares two values to determine inequality.

IIncrementOperators<UInt16>.CheckedIncrement(UInt16)	Increments a value.
IIncrementOperators<UInt16>.Increment(UInt16)	Increments a value.
IMinMaxValue<UInt16>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<UInt16>.MinValue	Gets the minimum value of the current type.
IModulus Operators<UInt16,UInt16,UInt16>.Modulus(UInt16, UInt16)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<UInt16,UInt16>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<UInt16,UInt16,UInt16>.Checked Multiply(UInt16, UInt16)	Multiplies two values together to compute their product.
IMultiplyOperators<UInt16,UInt16,UInt16>.Multiply(UInt16, UInt16)	Multiplies two values together to compute their product.
INumber<UInt16>.CopySign(UInt16, UInt16)	Copies the sign of a value to the sign of another value.
INumber<UInt16>.MaxNumber(UInt16, UInt16)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<UInt16>.MinNumber(UInt16, UInt16)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<UInt16>.Abs(UInt16)	Computes the absolute of a value.
INumberBase<UInt16>.IsCanonical(UInt16)	Determines if a value is in its canonical representation.
INumberBase<UInt16>.IsComplexNumber(UInt16)	Determines if a value represents a complex number.
INumberBase<UInt16>.IsFinite(UInt16)	Determines if a value is finite.
INumberBase<UInt16>.IsImaginaryNumber(UInt16)	Determines if a value represents a pure imaginary number.
INumberBase<UInt16>.IsInfinity(UInt16)	Determines if a value is infinite.
INumberBase<UInt16>.IsInteger(UInt16)	Determines if a value represents an integral number.

INumberBase<UInt16>.IsNaN(UInt16)	Determines if a value is NaN.
INumberBase<UInt16>.IsNegative(UInt16)	Determines if a value is negative.
INumberBase<UInt16>.IsNegativeInfinity(UInt16)	Determines if a value is negative infinity.
INumberBase<UInt16>.IsNormal(UInt16)	Determines if a value is normal.
INumberBase<UInt16>.IsPositive(UInt16)	Determines if a value is positive.
INumberBase<UInt16>.IsPositiveInfinity(UInt16)	Determines if a value is positive infinity.
INumberBase<UInt16>.IsRealNumber(UInt16)	Determines if a value represents a real number.
INumberBase<UInt16>.IsSubnormal(UInt16)	Determines if a value is subnormal.
INumberBase<UInt16>.IsZero(UInt16)	Determines if a value is zero.
INumberBase<UInt16>.MaxMagnitude(UInt16, UInt16)	Compares two values to compute which is greater.
INumberBase<UInt16>.MaxMagnitudeNumber(UInt16, UInt16)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt16>.MinMagnitude(UInt16, UInt16)	Compares two values to compute which is lesser.
INumberBase<UInt16>.MinMagnitudeNumber(UInt16, UInt16)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt16>.One	Gets the value `1` for the type.
INumberBase<UInt16>.Radix	Gets the radix, or base, for the type.
INumberBase<UInt16>.TryConvertFromChecked<TOther>(TOther, UInt16)	
INumberBase<UInt16>.TryConvertFromSaturating<TOther>(TOther, UInt16)	
INumberBase<UInt16>.TryConvertFromTruncating<TOther>(TOther, UInt16)	
INumberBase<UInt16>.TryConvertToChecked<TOther>(UInt16, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the

	representable range of the current type.
INumberBase<UInt16>.TryConvertToSaturating<TOther>(UInt16, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<UInt16>.TryConvertToTruncating<TOther>(UInt16, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<UInt16>.Zero	Gets the value `0` for the type.
IShiftOperators<UInt16,Int32,UInt16>.LeftShift(UInt16, Int32)	Shifts a value left by a given amount.
IShiftOperators<UInt16,Int32,UInt16>.RightShift(UInt16, Int32)	Shifts a value right by a given amount.
IShiftOperators<UInt16,Int32,UInt16>.UnsignedRightShift(UInt16, Int32)	Shifts a value right by a given amount.
ISubtractionOperators<UInt16,UInt16,UInt16>.CheckedSubtraction(UInt16, UInt16)	Subtracts two values to compute their difference.
ISubtractionOperators<UInt16,UInt16,UInt16>.Subtraction(UInt16, UInt16)	Subtracts two values to compute their difference.
IUnaryNegationOperators<UInt16,UInt16>.CheckedUnaryNegation(UInt16)	Computes the checked unary negation of a value.
IUnaryNegationOperators<UInt16,UInt16>.UnaryNegation(UInt16)	Computes the unary negation of a value.
IUnaryPlusOperators<UInt16,UInt16>.UnaryPlus(UInt16)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Int32
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of UInt16. This field is constant.

C#

```
public const ushort MaxValue = 65535;
```

Field Value

***Value* = 65535**

UInt16

Examples

The following example uses the UInt16.MaxValue and UInt16.MinValue properties to ensure that an Int32 value is in the range of the UInt16 type before converting it to a UInt16 value. This prevents the conversion operation from throwing an OverflowException if the integer value is not in the range of the UInt16 type.

C#

```
int integerValue = 1216;
ushort uIntegerValue;

if (integerValue >= ushort.MinValue & integerValue <= ushort.MaxValue)
{
   uIntegerValue = (ushort) integerValue;
   Console.WriteLine(uIntegerValue);
}
else
{
   Console.WriteLine("Unable to convert {0} to a UInt16t.", integerValue);
}
```

Remarks

The value of this constant is 65535; that is, hexadecimal 0xFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of UInt16. This field is constant.

C#

```
public const ushort MinValue = 0;
```

Field Value

Value* = *0

UInt16

Examples

The following example uses the MinValue and MaxValue properties to ensure that an Int32 value is in the range of the UInt16 type before converting it to a UInt16 value. This prevents the conversion operation from throwing an OverflowException if the integer value is not in the range of the UInt16 type.

C#

```
int integerValue = 1216;
ushort uIntegerValue;

if (integerValue >= ushort.MinValue & integerValue <= ushort.MaxValue)
{
   uIntegerValue = (ushort) integerValue;
   Console.WriteLine(uIntegerValue);
}
else
{
   Console.WriteLine("Unable to convert {0} to a UInt16t.", integerValue);
}
```

Remarks

The value of this constant is 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.Clamp(UInt16, UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static ushort Clamp (ushort value, ushort min, ushort max);
```

Parameters

`value` UInt16

The value to clamp.

`min` UInt16

The inclusive minimum to which `value` should clamp.

`max` UInt16

The inclusive maximum to which `value` should clamp.

Returns

UInt16

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or UInt16 and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CompareTo(UInt16)	Compares this instance to a specified 16-bit unsigned integer and returns an indication of their relative values.

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

value Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a UInt16.

Examples

The following example demonstrates the CompareTo method.

C#

```
public class Temperature : IComparable {
    /// <summary>
    /// IComparable.CompareTo implementation.
    /// </summary>
    public int CompareTo(object obj) {
        if(obj is Temperature) {
            Temperature temp = (Temperature) obj;

            return m_value.CompareTo(temp.m_value);
        }

        throw new ArgumentException("object is not a Temperature");
    }

    // The value holder
    protected ushort m_value;

    public ushort Value {
        get {
            return m_value;
        }
```

```
        set {
            m_value = value;
        }
    }
}
```

Remarks

Any instance of UInt16, regardless of its value, is considered greater than `null`.

`value` must be `null` or an instance of UInt16; otherwise, an exception is thrown.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CompareTo(UInt16)

Compares this instance to a specified 16-bit unsigned integer and returns an indication of their relative values.

C#

```
public int CompareTo (ushort value);
```

Parameters

`value` UInt16

An unsigned integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
```

```
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
```

```
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
```

```
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the UInt16.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static ushort CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt16

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static ushort CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

TOther

The type of `value`.

Parameters

value TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt16

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static ushort CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt16

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.DivRem(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

C#

```
public static (ushort Quotient, ushort Remainder) DivRem (ushort left,
ushort right);
```

Parameters

`left` UInt16

The value which `right` divides.

`right` UInt16

The value which divides `left`.

Returns

ValueTuple<UInt16,UInt16>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or UInt16.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt16)	Returns a value indicating whether this instance is equal to a specified UInt16 value.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare to this instance.

Returns

Boolean

`true` if `obj` is an instance of UInt16 and equals the value of this instance; otherwise, `false`.

Examples

The following example demonstrates the Equals method.

C#

```
UInt16 myVariable1 = 10;
UInt16 myVariable2 = 10;

//Display the declaring type.
Console.WriteLine("\nType of 'myVariable1' is '{0}' and" +
     " value is :{1}", myVariable1.GetType(), myVariable1);
Console.WriteLine("Type of 'myVariable2' is '{0}' and" +
     " value is :{1}", myVariable2.GetType(), myVariable2);

// Compare 'myVariable1' instance with 'myVariable2' Object.
if (myVariable1.Equals(myVariable2))
    Console.WriteLine("\nStructures 'myVariable1' and " +
          "'myVariable2' are equal");
else
    Console.WriteLine("\nStructures 'myVariable1' and " +
          "'myVariable2' are not equal");
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt16) method overloads. If an implicit conversion between the `obj` argument and a UInt16 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt16) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a UInt16 value. The following example illustrates the difference in behavior between the two method overloads. In the case of a Byte value, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt16) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static ushort value = 112;

   public static void Main()
   {
      byte byte1= 112;
```

```
        Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
        TestObjectForEquality(byte1);

        short short1 = 112;
        Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
        TestObjectForEquality(short1);

        int int1 = 112;
        Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
        TestObjectForEquality(int1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (UInt16) = 112 (Byte): False
//
//       value = short1:             False
//       112 (UInt16) = 112 (Int16): False
//
//       value = int1:               False
//       112 (UInt16) = 112 (Int32): False
//
//       value = sbyte1:             False
//       112 (UInt16) = 112 (SByte): False
//
//       value = dec1:                 False
//       112 (UInt16) = 112 (Decimal): False
//
//       value = dbl1:                 False
//       112 (UInt16) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(UInt16)

Returns a value indicating whether this instance is equal to a specified UInt16 value.

C#

```
public bool Equals (ushort obj);
```

Parameters

`obj` UInt16

A 16-bit unsigned integer to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt16) method overloads. If an implicit conversion between the `obj` argument and a UInt16 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt16) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a UInt16 value. The following example illustrates the difference in behavior between the two method overloads. In the case of a Byte value, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt16) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static ushort value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
      TestObjectForEquality(int1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      decimal dec1 = 112m;
      Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
      TestObjectForEquality(dec1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }

   private static void TestObjectForEquality(Object obj)
   {
```

```
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (UInt16) = 112 (Byte): False
//
//       value = short1:             False
//       112 (UInt16) = 112 (Int16): False
//
//       value = int1:               False
//       112 (UInt16) = 112 (Int32): False
//
//       value = sbyte1:             False
//       112 (UInt16) = 112 (SByte): False
//
//       value = dec1:                 False
//       112 (UInt16) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (UInt16) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UInt16.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt16.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type UInt16.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, UInt16.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IsEvenInteger(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IsOddInteger(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IsPow2(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.LeadingZeroCount(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static ushort LeadingZeroCount (ushort value);
```

Parameters

`value` UInt16

The value whose leading zeroes are to be counted.

Returns

UInt16

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Log2(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static ushort Log2 (ushort value);
```

Parameters

`value` UInt16

The value whose log2 is to be computed.

Returns

UInt16

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Max(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static ushort Max (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Min(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static ushort Min (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 16-bit unsigned integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 16-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 16-bit unsigned integer equivalent. This method is not CLS-compliant. The CLS-compliant alternative is Parse(String, NumberStyles).
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Converts the string representation of a number to its 16-bit unsigned integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int32.Parse(String)**

Converts the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent.

C#

```
public static ushort Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt16

A 16-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number that is less than UInt16.MinValue or greater than UInt16.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example uses the Parse(String, NumberStyles, IFormatProvider) method to convert various string representations of numbers to 16-bit unsigned integer values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] cultureNames = { "en-US", "fr-FR" };
      NumberStyles[] styles= { NumberStyles.Integer,
                               NumberStyles.Integer |
NumberStyles.AllowDecimalPoint };
      string[] values = { "1702", "+1702.0", "+1702,0", "-1032.00",
                          "-1032,00", "1045.1", "1045,1" };

      // Parse strings using each culture
      foreach (string cultureName in cultureNames)
      {
```

```
         CultureInfo ci = new CultureInfo(cultureName);
         Console.WriteLine("Parsing strings using the {0} culture",
                           ci.DisplayName);
         // Use each style.
         foreach (NumberStyles style in styles)
         {
            Console.WriteLine("   Style: {0}", style.ToString());
            // Parse each numeric string.
            foreach (string value in values)
            {
               try {
                  Console.WriteLine("      Converted '{0}' to {1}.",
value,
                                    UInt16.Parse(value, style, ci));
               }
               catch (FormatException) {
                  Console.WriteLine("      Unable to parse '{0}'.",
value);
               }
               catch (OverflowException) {
                  Console.WriteLine("      '{0}' is out of range of the
UInt16 type.",
                                    value);
               }
            }
         }
      }
   }
}
// The example displays the following output:
//       Parsing strings using the English (United States) culture
//          Style: Integer
//             Converted '1702' to 1702.
//             Unable to parse '+1702.0'.
//             Unable to parse '+1702,0'.
//             Unable to parse '-1032.00'.
//             Unable to parse '-1032,00'.
//             Unable to parse '1045.1'.
//             Unable to parse '1045,1'.
//          Style: Integer, AllowDecimalPoint
//             Converted '1702' to 1702.
//             Converted '+1702.0' to 1702.
//             Unable to parse '+1702,0'.
//             '-1032.00' is out of range of the UInt16 type.
//             Unable to parse '-1032,00'.
//             '1045.1' is out of range of the UInt16 type.
//             Unable to parse '1045,1'.
//       Parsing strings using the French (France) culture
//          Style: Integer
//             Converted '1702' to 1702.
//             Unable to parse '+1702.0'.
//             Unable to parse '+1702,0'.
//             Unable to parse '-1032.00'.
//             Unable to parse '-1032,00'.
//             Unable to parse '1045.1'.
```

```
//             Unable to parse '1045,1'.
//          Style: Integer, AllowDecimalPoint
//             Converted '1702' to 1702.
//             Unable to parse '+1702.0'.
//             Converted '+1702,0' to 1702.
//             Unable to parse '-1032.00'.
//             '-1032,00' is out of range of the UInt16 type.
//             Unable to parse '1045.1'.
//             '1045,1' is out of range of the UInt16 type.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive or negative sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object that is returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. (The method throws an OverflowException if `s` includes a negative sign and represents a non-zero number.) The sign can appear

Element	Description
	at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0-9 if `style` includes the NumberStyles.AllowExponent flag, or one or more occurrences of the digit 0 if it does not. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
NumberStyles.None	Decimal digits only.

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, if style does not include the NumberStyles.AllowExponent flag, *fractional_digits* must consist of only one or more 0 digits; otherwise, an OverflowException is thrown.
NumberStyles.AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign before *digits*.
NumberStyles.AllowTrailingSign	A sign after *digits*.
NumberStyles.AllowParentheses	Parentheses before and after *digits* to indicate a negative value.
NumberStyles.AllowThousands	The group separator (,) element.
NumberStyles.AllowCurrencySymbol	The currency ($) element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal digits are 0 through 9, a through f, and A through F. A prefix, such as "0x", is not supported and causes the parse operation to fail. The only other flags that can be combined with NumberStyles.AllowHexSpecifier are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If the `s` parameter is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the parse operation to throw an exception.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent.

C#

```
public static ushort Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicate the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt16

A 16-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static ushort Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt16

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int32.Parse(String)**

Converts the string representation of a number in a specified culture-specific format to its 16-bit unsigned integer equivalent.

C#

```
public static ushort Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt16

A 16-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example instantiates a custom culture that uses two plus signs (++) as its positive sign. It then calls the Parse(String, IFormatProvider) method to parse an array of strings by using CultureInfo objects that represent both this custom culture and the invariant culture.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define a custom culture that uses "++" as a positive sign.
      CultureInfo ci = new CultureInfo("");
      ci.NumberFormat.PositiveSign = "++";
      // Create an array of cultures.
      CultureInfo[] cultures = { ci, CultureInfo.InvariantCulture };
      // Create an array of strings to parse.
      string[] values = { "++1403", "-0", "+0", "+16034",
                          Int16.MinValue.ToString(), "14.0", "18012" };
      // Parse the strings using each culture.
      foreach (CultureInfo culture in cultures)
      {
         Console.WriteLine("Parsing with the '{0}' culture.",
culture.Name);
         foreach (string value in values)
         {
            try {
               ushort number = UInt16.Parse(value, culture);
               Console.WriteLine("   Converted '{0}' to {1}.", value,
number);
            }
            catch (FormatException) {
               Console.WriteLine("   The format of '{0}' is invalid.",
value);
            }
            catch (OverflowException) {
               Console.WriteLine("   '{0}' is outside the range of a
UInt16 value.", value);
            }
         }
      }
   }
}
```

```
// The example displays the following output:
//       Parsing with the  culture.
//          Converted '++1403' to 1403.
//          Converted '-0' to 0.
//          The format of '+0' is invalid.
//          The format of '+16034' is invalid.
//          '-32768' is outside the range of a UInt16 value.
//          The format of '14.0' is invalid.
//          Converted '18012' to 18012.
//       Parsing with the '' culture.
//          The format of '++1403' is invalid.
//          Converted '-0' to 0.
//          Converted '+0' to 0.
//          Converted '+16034' to 16034.
//          '-32768' is outside the range of a UInt16 value.
//          The format of '14.0' is invalid.
//          Converted '18012' to 18012.
```

Remarks

The `s` parameter contains a number of the form:

[*ws*][*sign*]*digits*[*ws*]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign, or a negative sign if `s` represents the value zero.
digits	A sequence of digits ranging from 0 to 9.

The s parameter is interpreted using the NumberStyles.Integer style. In addition to the byte value's decimal digits, only leading and trailing spaces along with a leading sign is allowed. (If the negative sign is present, `s` must represent a value of zero or the method throws an OverflowException.) To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific

information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String, NumberStyles)

> Important
>
> This API is not CLS-compliant.

Converts the string representation of a number in a specified style to its 16-bit unsigned integer equivalent.

This method is not CLS-compliant. The CLS-compliant alternative is Parse(String, NumberStyles).

```
C#

public static ushort Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

s String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

style NumberStyles

A bitwise combination of the enumeration values that specify the permitted format of `s`. A typical value to specify is Integer.

Returns

UInt16

A 16-bit unsigned integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number less than UInt16.MinValue or greater than UInt16.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example tries to parse each element in a string array by using a number of NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] values = { " 214 ", "1,064", "(0)", "1241+", " + 214 ", "
+214 ", "2153.0", "1e03", "1300.0e-2" };
      NumberStyles whitespace =  NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite;
      NumberStyles[] styles = { NumberStyles.None, whitespace,
                                NumberStyles.AllowLeadingSign |
NumberStyles.AllowTrailingSign | whitespace,
                                NumberStyles.AllowThousands |
NumberStyles.AllowCurrencySymbol,
                                NumberStyles.AllowExponent |
NumberStyles.AllowDecimalPoint };

      // Attempt to convert each number using each style combination.
      foreach (string value in values)
      {
         Console.WriteLine("Attempting to convert '{0}':", value);
         foreach (NumberStyles style in styles)
         {
            try {
               ushort number = UInt16.Parse(value, style);
               Console.WriteLine("   {0}: {1}", style, number);
            }
            catch (FormatException) {
               Console.WriteLine("   {0}: Bad Format", style);
            }
         }
         Console.WriteLine();
      }
   }
}
```

```
// The example display the following output:
//    Attempting to convert ' 214 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: 214
//       Integer, AllowTrailingSign: 214
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '1,064':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: 1064
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '(0)':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '1241+':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 1241
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' + 214 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' +214 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 214
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '2153.0':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 2153
//
//    Attempting to convert '1e03':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
```

```
//       AllowDecimalPoint, AllowExponent: 1000
//
//    Attempting to convert '1300.0e-2':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 13
```

Remarks

The `style` parameter defines the style elements (such as white space, the positive or negative sign symbol, the group separator symbol, or the decimal point symbol) that are allowed in the `s` parameter for the parse operation to succeed. `style` must be a combination of bit flags from the NumberStyles enumeration. The `style` parameter makes this method overload useful when `s` contains the string representation of a hexadecimal value, when the number system (decimal or hexadecimal) represented by `s` is known only at run time, or when you want to disallow white space or a sign symbol in `s`.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may contain the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current

Element	Description
	culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific group separator symbol. The current culture's group separator can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, a FormatException is thrown.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the UInt16 type. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

`NumberStyles` value	Elements permitted in `s` in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the start of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

Unlike the other NumberStyles values, which allow for, but do not require, the presence of particular style elements in `s`, the NumberStyles.AllowHexSpecifier style value means that the individual numeric characters in `s` are always interpreted as hexadecimal characters. Valid hexadecimal characters are 0-9, A-F, and a-f. A prefix, such as "0x", is not supported and causes the parse operation to fail. The only other flags that can be combined with the `style` parameter are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

Note

If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static ushort Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt16

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static ushort Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt16

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int32.Parse(String)**

Converts the string representation of a number to its 16-bit unsigned integer equivalent.

C#

```
public static ushort Parse (string s);
```

Parameters

s String

A string that represents the number to convert.

Returns

UInt16

A 16-bit unsigned integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than UInt16.MinValue or greater than UInt16.MaxValue.

Examples

The following example calls the Parse(String) method to convert each element in a string array to an unsigned 16-bit integer.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string[] values = { "-0", "17", "-12", "185", "66012", "+0",
                          "", null, "16.1", "28.0", "1,034" };
      foreach (string value in values)
      {
         try {
            ushort number = UInt16.Parse(value);
            Console.WriteLine("'{0}' --> {1}", value, number);
         }
         catch (FormatException) {
            Console.WriteLine("'{0}' --> Bad Format", value);
         }
         catch (OverflowException) {
            Console.WriteLine("'{0}' --> OverflowException", value);
         }
         catch (ArgumentNullException) {
            Console.WriteLine("'{0}' --> Null", value);
         }
      }
   }
}
// The example displays the following output:
//       '-0' --> 0
```

```
//       '17' --> 17
//       '-12' --> OverflowException
//       '185' --> 185
//       '66012' --> OverflowException
//       '+0' --> 0
//       '' --> Bad Format
//       '' --> Null
//       '16.1' --> Bad Format
//       '28.0' --> Bad Format
//       '1,034' --> Bad Format
```

Remarks

The `s` parameter should be the string representation of a number in the following form.

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits	A sequence of digits ranging from 0 to 9. Any leading zeros are ignored.

> **Note**
>
> The string specified by the `s` parameter is interpreted by using the **NumberStyles.Integer** style. It cannot contain any group separators or decimal separator, and it cannot have a decimal portion.

The `s` parameter is parsed by using the formatting information in a System.Globalization.NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string by using the formatting information of a specific culture, use the Parse(String, IFormatProvider) method.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.PopCount(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static ushort PopCount (ushort value);
```

Parameters

`value` UInt16

The value whose set bits are to be counted.

Returns

UInt16

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.RotateLeft(UInt16, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

C#

```
public static ushort RotateLeft (ushort value, int rotateAmount);
```

Parameters

`value` UInt16

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

UInt16

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.RotateRight(UInt16, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
public static ushort RotateRight (ushort value, int rotateAmount);
```

Parameters

`value` UInt16

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

UInt16

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.Sign(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (ushort value);
```

Parameters

`value` UInt16

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString()	Converts the numeric value of this instance to its equivalent string representation.

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance, which consists of a sequence of digits ranging from 0 to 9, without a sign or leading zeros.

Implements

ToString(IFormatProvider)

Examples

The following example formats a 16-bit signed integer value by using several format providers, including one for the invariant culture. The output from the example illustrates that the formatted string returned by the ToString(IFormatProvider) method is the same regardless of the format provider.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define an array of CultureInfo objects.
      CultureInfo[] ci = { new CultureInfo("en-US"),
                           new CultureInfo("fr-FR"),
                           CultureInfo.InvariantCulture };
      UInt16 value = 18924;
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        GetName(ci[0]), GetName(ci[1]), GetName(ci[2]));
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        value.ToString(ci[0]), value.ToString(ci[1]),
value.ToString(ci[2]));
   }

   private static string GetName(CultureInfo ci)
   {
      if (ci.Equals(CultureInfo.InvariantCulture))
         return "Invariant";
      else
         return ci.Name;
   }
}
```

```
// The example displays the following output:
//          en-US          fr-FR      Invariant
//          18924          18924          18924
```

Remarks

The ToString(IFormatProvider) method formats a UInt16 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific formatting information. However, none of the properties of the NumberFormatInfo are used when formatting with the general numeric format specifier ("G").

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

The `format` parameter is invalid.

Examples

The following example displays a 16-bit unsigned integer value by using each standard format string and some custom format strings.

```
C#

using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
        ushort value = 21708;
        string[] specifiers = { "G", "C", "D3", "E2", "e3", "F",
                                "N", "P", "X", "000000.0", "#.0",
                                "00000000;(0);**Zero**" };

        foreach (string specifier in specifiers)
           Console.WriteLine("{0}: {1}", specifier,
value.ToString(specifier));
    }
}
// The example displays the following output:
//       G: 21708
//       C: $21,708.00
//       D3: 21708
//       E2: 2.17E+004
//       e3: 2.171e+004
//       F: 21708.00
//       N: 21,708.00
//       P: 2,170,800.00 %
//       X: 54CC
//       000000.0: 021708.0
//       #.0: 21708.0
//       00000000;(0);**Zero**: 00021708
```

Remarks

The ToString(String) method formats a UInt16 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid standard numeric format specifier, or any combination of custom numeric format specifiers. If `format` is equal to String.Empty or is `null`, the return value of the current UInt16 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The format of the returned string is determined by the NumberFormatInfo object for the current culture. Depending on the `format` parameter, this object controls symbols such as the group separator and the decimal point symbol in the output string. To provide formatting information for cultures other than the current culture, call the ToString(String, IFormatProvider) overload.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

```
C#
```

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance, as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid.

Examples

The following example displays a 16-bit unsigned integer value by using the standard numeric format specifiers and a number of specific CultureInfo objects.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define cultures whose formatting conventions are to be used.
      CultureInfo[] cultures = { CultureInfo.CreateSpecificCulture("en-
US"),
```

```
                                    CultureInfo.CreateSpecificCulture("fr-
FR"),
                                    CultureInfo.CreateSpecificCulture("es-
ES") };
      string[] specifiers = {"G", "C", "D4", "E2", "F", "N", "P", "X2"};
      ushort value = 22042;

      foreach (string specifier in specifiers)
      {
         foreach (CultureInfo culture in cultures)
            Console.WriteLine("{0,2} format using {1} culture: {2, 16}",
                              specifier, culture.Name,
                              value.ToString(specifier, culture));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//        G format using en-US culture:            22042
//        G format using fr-FR culture:            22042
//        G format using es-ES culture:            22042
//
//        C format using en-US culture:       $22,042.00
//        C format using fr-FR culture:      22 042,00 €
//        C format using es-ES culture:      22.042,00 €
//
//       D4 format using en-US culture:            22042
//       D4 format using fr-FR culture:            22042
//       D4 format using es-ES culture:            22042
//
//       E2 format using en-US culture:        2.20E+004
//       E2 format using fr-FR culture:        2,20E+004
//       E2 format using es-ES culture:        2,20E+004
//
//        F format using en-US culture:         22042.00
//        F format using fr-FR culture:         22042,00
//        F format using es-ES culture:         22042,00
//
//        N format using en-US culture:        22,042.00
//        N format using fr-FR culture:        22 042,00
//        N format using es-ES culture:        22.042,00
//
//        P format using en-US culture:   2,204,200.00 %
//        P format using fr-FR culture:   2 204 200,00 %
//        P format using es-ES culture:   2.204.200,00 %
//
//       X2 format using en-US culture:             561A
//       X2 format using fr-FR culture:             561A
//       X2 format using es-ES culture:             561A
```

Remarks

The ToString(String, IFormatProvider) method formats a UInt16 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be any valid Standard Numeric Format Strings, or any combination of Custom Numeric Format Strings. If `format` is equal to String.Empty or is `null`, the return value of the current UInt16 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string returned by this method. When the ToString(String, IFormatProvider) method is invoked, it calls the `provider` parameter's IFormatProvider.GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information for formatting the current UInt16 value, such as the group separator symbol or the decimal point symbol. There are three ways to use the `provider` parameter to supply formatting information to the ToString(String, IFormatProvider) method:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.

- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the formatting of the returned string is based on the NumberFormatInfo object of the current culture.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, which consists of a sequence of digits ranging from 0 to 9, without a sign or leading zeros.

Examples

The following example displays a UInt16 value by using the default ToString() method. It also displays the string representations of the UInt16 value that results from using some standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      ushort value = 16324;
      // Display value using default ToString method.
      Console.WriteLine(value.ToString());
      Console.WriteLine();

      // Define an array of format specifiers.
      string[] formats = { "G", "C", "D", "F", "N", "X" };
      // Display value using the standard format specifiers.
      foreach (string format in formats)
         Console.WriteLine("{0} format specifier: {1,12}",
                           format, value.ToString(format));
   }
}
// The example displays the following output:
//       16324
//
//       G format specifier:        16324
//       C format specifier:   $16,324.00
//       D format specifier:        16324
//       F format specifier:     16324.00
//       N format specifier:    16,324.00
//       X format specifier:         3FC4
```

Remarks

The ToString() method formats a UInt16 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UInt16.TrailingZeroCount(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static ushort TrailingZeroCount (ushort value);
```

Parameters

`value` UInt16

The value whose trailing zeroes are to be counted.

Returns

UInt16

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current unsigned short number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current unsigned short number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, UInt16)	Tries to convert the span representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt16)	Tries to convert the string representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt16)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, UInt16)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt16)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt16)	Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt16)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(String, NumberStyles, IFormatProvider, UInt16)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt16)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out ushort result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt16

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, UInt16)

> Important
>
> This API is not CLS-compliant.

Tries to convert the span representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out ushort result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters representing the number to convert.

`result` UInt16

When this method returns, contains the 16-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in the correct format. , or represents a number less than UInt16.MinValue or greater than UInt16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, UInt16)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int32.TryParse(String, Int32)**

Tries to convert the string representation of a number to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (string? s, out ushort result);
```

Parameters

`s` String

A string that represents the number to convert.

`result` UInt16

When this method returns, contains the 16-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in the correct format, or represents a number less than UInt16.MinValue or greater than UInt16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParse(String, UInt16) method once for each element in a string array.

```
C#
```

```
string[] numericStrings = { "1293.8", "+1671.7", "28347.",
                            "   33113684  ", "(0)", "-0", "-1",
                            "+1293617", "18-", "119870", "31,024",
                            "  3127094 ", "00700000" };
uint number;
foreach (string numericString in numericStrings)
{
   if (UInt32.TryParse(numericString, out number))
      Console.WriteLine("Converted '{0}' to {1}.", numericString,
number);
   else
      Console.WriteLine("Cannot convert '{0}' to a UInt32.",
numericString);
}
// The example displays the following output:
//       Cannot convert '1293.8' to a UInt32.
//       Cannot convert '+1671.7' to a UInt32.
//       Cannot convert '28347.' to a UInt32.
//       Converted '   33113684  ' to 33113684.
//       Cannot convert '(0)' to a UInt32.
//       Converted '-0' to 0.
//       Cannot convert '-1' to a UInt32.
//       Converted '+1293617' to 1293617.
//       Cannot convert '18-' to a UInt32.
//       Converted '119870' to 119870.
//       Cannot convert '31,024' to a UInt32.
//       Converted '  3127094 ' to 3127094.
//       Converted '0070000' to 70000.
```

Remarks

The TryParse(String, UInt16) method is like the Parse(String) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be successfully parsed.

The `s` parameter should be the string representation of a decimal number in the following form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture.
digits	A sequence of decimal digits ranging from 0 to 9.

> **Note**
>
> The string specified by the `s` parameter cannot contain any group separators or decimal separator, and it cannot have a decimal portion.

The `s` parameter is interpreted by using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces with a leading sign are allowed. To explicitly define the style elements with the culture-specific formatting information that can be present in `s`, call the TryParse(String, NumberStyles, IFormatProvider, UInt16) method.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object for the current system culture. For more information, see NumberFormatInfo.CurrentInfo.

This overload interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the TryParse(String, NumberStyles, IFormatProvider, UInt16) overload instead.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt16)

Tries to parse a span of characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out ushort result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt16

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, UInt16)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
ushort result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt16

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt16)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
ushort result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt16

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, UInt16)

Tries to convert a UTF-8 character span containing the string representation of a number to its 16-bit unsigned integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out ushort
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` UInt16

When this method returns, contains the 16-bit unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt16)

Important

This API is not CLS-compliant.

Tries to convert the span representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
ushort result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt16

When this method returns, contains the 16-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than UInt16.MinValue or greater than UInt16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, UInt16)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int32.TryParse(String, Int32)**

Tries to convert the string representation of a number in a specified style and culture-specific format to its 16-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out ushort result);
```

Parameters

`s` String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt16

When this method returns, contains the 16-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than UInt16.MinValue or greater than UInt16.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, UInt16) method with a number of different strings and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "10603";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-10603";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);
```

```
      numericString = "29103.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "2210E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "9112E-01";
      CallTryParse(numericString, styles);

      numericString = "312E01";
      CallTryParse(numericString, styles);

      numericString = "FFC8";
      CallTryParse(numericString, NumberStyles.HexNumber);

      numericString = "0x8F8C";
      CallTryParse(numericString, NumberStyles.HexNumber);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      ushort number;
      bool result = UInt16.TryParse(stringToConvert, styles,
                                    CultureInfo.InvariantCulture, out
number);
      if (result)
         Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
      else
         Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
   }
}
// The example displays the following output:
//       Converted '10603' to 10603.
//       Attempted conversion of '-10603' failed.
//       Converted '29103.00' to 29103.
//       Attempted conversion of '10345.72' failed.
//       Converted '2210E-01' to 221.
//       Attempted conversion of '9112E-01' failed.
//       Converted '312E01' to 3120.
//       Converted 'FFC8' to 65480.
//       Attempted conversion of '0x8F8C' failed.
```

Remarks

The TryParse(String, NumberStyles, IFormatProvider, UInt16) method is like the Parse(String, NumberStyles, IFormatProvider) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Items in square brackets ([and]) are optional. Or, if the `style` parameter includes AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

ws Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.

$ A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.

sign An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, if the negative sign is present, `s` can only represent the value zero for the parse operation to succeed.

digits A sequence of digits from 0 through 9.

, A culture-specific group separator. The group separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.

. A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.

fractional_digits One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.

E The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.

exponential_digits A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.

hexdigits A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
None	Decimal digits only.
AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits, or the method returns `false`.
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*. If `s` represents a number in exponential notation, it cannot have a non-zero, fractional component.

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element before *digits*.
AllowTrailingSign	The *sign* element after *digits*.
AllowParentheses	The *sign* element in the form of parentheses enclosing a zero numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The *ws*, *sign*, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal digits are 0 through 9, a through f, and A through F. A prefix such as "0x" is not supported and causes the parse operation to fail. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, HexNumber, that includes both white-space flags.)

> **Note**
>
> If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific

information about the format of `s`. The `provider` parameter can be any one of the following:

- A CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- A NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- A custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The current value of this instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The current value of this instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The current value of this instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of this instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The current value of this instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSByte(IFormatProvider).

```
C#
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The current value of this instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSByte(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The current value pf this instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this UInt16 value.

provider IFormatProvider

An IFormatProvider implementation that supplies information about the format of the returned value.

Returns

Object

The current value of this instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.To UInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The current value of this instance, unchanged.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The current value of this instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt32(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The current value of this instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt16 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt64(UInt16) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.IAdditionOperators<UInt16,UInt16,UInt16>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static ushort IAdditionOperators<ushort,ushort,ushort>.operator + (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to which `right` is added.

`right` UInt16

The value that's added to `left`.

Returns

UInt16

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IAdditionOperators<UInt16,UInt16,UInt16>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static ushort IAdditionOperators<ushort,ushort,ushort>.op_CheckedAddition
(ushort left, ushort right);
```

Parameters

`left` UInt16

The value to which `right` is added.

`right` UInt16

The value that's added to `left`.

Returns

UInt16

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IAdditiveIdentity<UInt16,UInt16>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static ushort
System.Numerics.IAdditiveIdentity<System.UInt16,System.UInt16>.AdditiveIdentity { get; }
```

Property Value

UInt16

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt16.IBinaryInteger<UInt16>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<ushort>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt16.IBinaryInteger<UInt16>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<ushort>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt16.IBinaryInteger<UInt16>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<ushort>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out ushort value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt16.IBinaryInteger<UInt16>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<ushort>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out ushort value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt16.IBinaryInteger<UInt16>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<ushort>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IBinaryInteger<UInt16>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<ushort>.TryWriteLittleEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IBinaryNumber<UInt16>.AllBits Set Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static ushort System.Numerics.IBinaryNumber<System.UInt16>.AllBitsSet { get;
}
```

Property Value

UInt16

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt16.IBitwiseOperators<UInt16,UInt16,UInt16>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static ushort IBitwiseOperators<ushort,ushort,ushort>.operator & (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to `and` with `right`.

`right` UInt16

The value to `and` with `left`.

Returns

UInt16

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IBitwiseOperators<UInt16,UInt16,UInt16>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static ushort IBitwiseOperators<ushort,ushort,ushort>.operator | (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to `or` with `right`.

`right` UInt16

The value to `or` with `left`.

Returns

UInt16

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IBitwiseOperators<UInt16,UInt16,UInt16>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static ushort IBitwiseOperators<ushort,ushort,ushort>.operator ^ (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to xor with `right`.

`right` UInt16

The value to xor with `left`.

Returns

UInt16

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IBitwiseOperators<UInt16,UInt16,UInt16>.OnesComplement Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#

static ushort IBitwiseOperators<ushort,ushort,ushort>.operator ~ (ushort
value);
```

Parameters

`value` UInt16

The value for which to compute the ones-complement.

Returns

UInt16

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IComparisonOperators<UInt16,UInt16,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<ushort,ushort,bool>.operator > (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IComparisonOperators<UInt16,UInt16,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<ushort,ushort,bool>.operator >= (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IComparisonOperators<UInt16,UInt16,Boolean>.LessThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<ushort,ushort,bool>.operator < (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IComparisonOperators<UInt16,UInt16,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<ushort,ushort,bool>.operator <= (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IDecrementOperators<UInt16>.CheckedDecrement(UInt16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#

static ushort IDecrementOperators<ushort>.op_CheckedDecrement (ushort
value);
```

Parameters

`value` UInt16

The value to decrement.

Returns

UInt16

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IDecrementOperators<UInt16>.Decrement(UInt16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static ushort IDecrementOperators<ushort>.operator -- (ushort value);
```

Parameters

`value` UInt16

The value to decrement.

Returns

UInt16

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IDivisionOperators<UInt16,UInt16,UInt16>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static ushort IDivisionOperators<ushort,ushort,ushort>.operator / (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value that `right` divides.

`right` UInt16

The value that divides `left`.

Returns

UInt16

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IEqualityOperators<UInt16,UInt16,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<ushort,ushort,bool>.operator == (ushort left,
ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IEqualityOperators<UInt16,UInt16,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<ushort,ushort,bool>.operator != (ushort left,
ushort right);
```

Parameters

`left` UInt16

The value to compare with `right`.

`right` UInt16

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IIncrementOperators<UInt16>.CheckedIncrement(UInt16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#

static ushort IIncrementOperators<ushort>.op_CheckedIncrement (ushort
value);
```

Parameters

`value` UInt16

The value to increment.

Returns

UInt16

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IIncrementOperators<UInt16>.Increment(UInt16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static ushort IIncrementOperators<ushort>.operator ++ (ushort value);
```

Parameters

`value` UInt16

The value to increment.

Returns

UInt16

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IMinMaxValue<UInt16>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static ushort System.Numerics.IMinMaxValue<System.UInt16>.MaxValue { get; }
```

Property Value

UInt16

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IMinMaxValue<UInt16>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static ushort System.Numerics.IMinMaxValue<System.UInt16>.MinValue { get; }
```

Property Value

UInt16

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IModulusOperators<UInt16,UInt16,UInt16>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static ushort IModulusOperators<ushort,ushort,ushort>.operator % (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value that `right` divides.

`right` UInt16

The value that divides `left`.

Returns

UInt16

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IMultiplicativeIdentity<UInt16,UInt16>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static ushort
System.Numerics.IMultiplicativeIdentity<System.UInt16,System.UInt16>.MultiplicativeIdentity { get; }
```

Property Value

UInt16

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt16.IMultiplyOperators<UInt16,UInt16,UInt16>.CheckedMultiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static ushort IMultiplyOperators<ushort,ushort,ushort>.op_CheckedMultiply
(ushort left, ushort right);
```

Parameters

`left` UInt16

The value that `right` multiplies.

`right` UInt16

The value that multiplies `left`.

Returns

UInt16

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IMultiplyOperators<UInt16,UInt16,UInt16>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static ushort IMultiplyOperators<ushort,ushort,ushort>.operator * (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value that `right` multiplies.

`right` UInt16

The value that multiplies `left`.

Returns

UInt16

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumber<UInt16>.CopySign(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static ushort INumber<ushort>.CopySign (ushort value, ushort sign);
```

Parameters

`value` UInt16

The value whose magnitude is used in the result.

`sign` UInt16

The value whose sign is used in the result.

Returns

UInt16

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumber<UInt16>.MaxNumber(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

C#

```
static ushort INumber<ushort>.MaxNumber (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumber<UInt16>.MinNumber(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

C#

```
static ushort INumber<ushort>.MinNumber (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.Abs(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static ushort INumberBase<ushort>.Abs (ushort value);
```

Parameters

`value` UInt16

The value for which to get its absolute.

Returns

UInt16

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsCanonical(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

C#

```
static bool INumberBase<ushort>.IsCanonical (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsComplexNumber(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

C#

```
static bool INumberBase<ushort>.IsComplexNumber (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsFinite(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<ushort>.IsFinite (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsImaginaryNumber(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<ushort>.IsImaginaryNumber (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsInfinity(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<ushort>.IsInfinity (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsInteger(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<ushort>.IsInteger (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsNaN(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<ushort>.IsNaN (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsNegative(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<ushort>.IsNegative (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsNegativeInfinity(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<ushort>.IsNegativeInfinity (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsNormal(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<ushort>.IsNormal (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsPositive(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<ushort>.IsPositive (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsPositiveInfinity(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<ushort>.IsPositiveInfinity (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsRealNumber(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<ushort>.IsRealNumber (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsSubnormal(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<ushort>.IsSubnormal (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.IsZero(UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<ushort>.IsZero (ushort value);
```

Parameters

`value` UInt16

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.MaxMagnitude(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static ushort INumberBase<ushort>.MaxMagnitude (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.MaxMagnitudeNumber(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static ushort INumberBase<ushort>.MaxMagnitudeNumber (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.MinMagnitude(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
static ushort INumberBase<ushort>.MinMagnitude (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.MinMagnitudeNumber(UInt16, UInt16) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static ushort INumberBase<ushort>.MinMagnitudeNumber (ushort x, ushort y);
```

Parameters

x UInt16

The value to compare with `y`.

y UInt16

The value to compare with `x`.

Returns

UInt16

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static ushort System.Numerics.INumberBase<System.UInt16>.One { get; }
```

Property Value

UInt16

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt16.INumberBase<UInt16>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.UInt16>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ushort>.TryConvertFromChecked<TOther> (TOther value,
out ushort result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ushort>.TryConvertFromSaturating<TOther> (TOther
value, out ushort result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ushort>.TryConvertFromTruncating<TOther> (TOther
value, out ushort result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt16

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<ushort>.TryConvertToChecked<TOther> (ushort value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<ushort>.TryConvertToSaturating<TOther> (ushort
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<ushort>.TryConvertToTruncating<TOther> (ushort
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt16

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.INumberBase<UInt16>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static ushort System.Numerics.INumberBase<System.UInt16>.Zero { get; }
```

Property Value

UInt16

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt16.IShift Operators<UInt16,Int32,UInt16>.Left Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static ushort IShiftOperators<ushort,int,ushort>.operator << (ushort value,
int shiftAmount);
```

Parameters

`value` UInt16

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

UInt16

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IShift Operators<UInt16,Int32,UInt16>.Right Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static ushort IShiftOperators<ushort,int,ushort>.operator >> (ushort value,
int shiftAmount);
```

Parameters

`value` UInt16

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt16

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IShift Operators<UInt16,Int32,UInt16>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static ushort IShiftOperators<ushort,int,ushort>.op_UnsignedRightShift
(ushort value, int shiftAmount);
```

Parameters

`value` UInt16

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt16

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.ISubtractionOperators<UInt16,UInt16,UInt16>.CheckedSubtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static ushort
ISubtractionOperators<ushort,ushort,ushort>.op_CheckedSubtraction (ushort
left, ushort right);
```

Parameters

`left` UInt16

The value from which `right` is subtracted.

`right` UInt16

The value that's subtracted from `left`.

Returns

UInt16

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.ISubtraction Operators<UInt16,UInt16,UInt16>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static ushort ISubtractionOperators<ushort,ushort,ushort>.operator - (ushort
left, ushort right);
```

Parameters

left UInt16

The value from which `right` is subtracted.

right UInt16

The value that's subtracted from `left`.

Returns

UInt16

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IUnaryNegationOperators<UInt16,UInt16>.CheckedUnaryNegation Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static ushort IUnaryNegationOperators<ushort,ushort>.op_CheckedUnaryNegation
(ushort value);
```

Parameters

`value` UInt16

The value for which to compute the unary negation.

Returns

UInt16

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IUnaryNegationOperators<UInt16,UInt16>.UnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#

static ushort IUnaryNegationOperators<ushort,ushort>.operator - (ushort
value);
```

Parameters

`value` UInt16

The value for which to compute the unary negation.

Returns

UInt16

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt16.IUnaryPlusOperators<UInt16,UInt16>.UnaryPlus(UInt16) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static ushort IUnaryPlusOperators<ushort,ushort>.operator + (ushort value);
```

Parameters

`value` UInt16

The value for which to compute the unary plus.

Returns

UInt16

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents a 32-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UInt32 : IComparable<uint>, IConvertible,
IEquatable<uint>, IParsable<uint>, ISpanParsable<uint>,
IUtf8SpanParsable<uint>, System.Numerics.IAdditionOperators<uint,uint,uint>,
System.Numerics.IAdditiveIdentity<uint,uint>,
System.Numerics.IBinaryInteger<uint>, System.Numerics.IBinaryNumber<uint>,
System.Numerics.IBitwiseOperators<uint,uint,uint>,
System.Numerics.IComparisonOperators<uint,uint,bool>,
System.Numerics.IDecrementOperators<uint>,
System.Numerics.IDivisionOperators<uint,uint,uint>,
System.Numerics.IEqualityOperators<uint,uint,bool>,
System.Numerics.IIncrementOperators<uint>,
System.Numerics.IMinMaxValue<uint>,
System.Numerics.IModulusOperators<uint,uint,uint>,
System.Numerics.IMultiplicativeIdentity<uint,uint>,
System.Numerics.IMultiplyOperators<uint,uint,uint>,
System.Numerics.INumber<uint>, System.Numerics.INumberBase<uint>,
System.Numerics.IShiftOperators<uint,int,uint>,
System.Numerics.ISubtractionOperators<uint,uint,uint>,
System.Numerics.IUnaryNegationOperators<uint,uint>,
System.Numerics.IUnaryPlusOperators<uint,uint>,
System.Numerics.IUnsignedNumber<uint>
```

Inheritance Object → ValueType → UInt32

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UInt32> , IConvertible , IEquatable<UInt32> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<UInt32> , IParsable<TSelf> , ISpanParsable<UInt32> , ISpanParsable<TSelf> , IAdditionOperators<UInt32,UInt32,UInt32> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UInt32,UInt32> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UInt32> , IBinaryNumber<UInt32> , IBinaryNumber<TSelf> , IBitwiseOperators<UInt32,UInt32,UInt32> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UInt32,UInt32,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UInt32> , IDecrementOperators<TSelf> , IDivisionOperators<UInt32,UInt32,UInt32> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UInt32,UInt32,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UInt32> , IIncrementOperators<TSelf> , IMinMaxValue<UInt32> , IModulusOperators<UInt32,UInt32,UInt32> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UInt32,UInt32> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UInt32,UInt32,UInt32> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UInt32> , INumber<TSelf> , INumberBase<UInt32> , INumberBase<TSelf> , IShiftOperators<UInt32,Int32,UInt32> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UInt32,UInt32,UInt32> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UInt32,UInt32> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<UInt32,UInt32> , IUnaryPlusOperators<TSelf,TSelf> ,

IUnsignedNumber<UInt32> , IUtf8SpanFormattable ,
IUtf8SpanParsable<UInt32> , IUtf8SpanParsable<TSelf>

Remarks

The `UInt32` value type represents unsigned integers with values ranging from 0 to 4,294,967,295.

> **Important**
>
> The **UInt32** type is not CLS-compliant. The CLS-compliant alternative type is **Int64**. **Int32** can be used instead to replace a **UInt32** value that ranges from zero to **MaxValue**. For more information about CLS compliance, see **Language Independence and Language-Independent Components**.

`UInt32` provides methods to compare instances of this type, convert the value of an instance to its String representation, and convert the String representation of a number to an instance of this type.

For information about how format specification codes control the String representation of value types, see Formatting Types.

Fields

Expand table

MaxValue	Represents the largest possible value of UInt32. This field is constant.
MinValue	Represents the smallest possible value of UInt32. This field is constant.

Methods

Expand table

Clamp(UInt32, UInt32, UInt32)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.

CompareTo(UInt32)	Compares this instance to a specified 32-bit unsigned integer and returns an indication of their relative values.
CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(UInt32, UInt32)	Computes the quotient and remainder of two values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt32)	Returns a value indicating whether this instance is equal to a specified UInt32.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type UInt32.
IsEvenInteger(UInt32)	Determines if a value represents an even integral number.
IsOddInteger(UInt32)	Determines if a value represents an odd integral number.
IsPow2(UInt32)	Determines if a value is a power of two.
LeadingZeroCount(UInt32)	Computes the number of leading zeros in a value.
Log2(UInt32)	Computes the log2 of a value.
Max(UInt32, UInt32)	Compares two values to compute which is greater.
Min(UInt32, UInt32)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned

	integer equivalent.
Parse(String)	Converts the string representation of a number to its 32-bit unsigned integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 32-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 32-bit unsigned integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent.
PopCount(UInt32)	Computes the number of bits that are set in a value.
RotateLeft(UInt32, Int32)	Rotates a value left by a given amount.
RotateRight(UInt32, Int32)	Rotates a value right by a given amount.
Sign(UInt32)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(UInt32)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current unsigned integer number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider,	Tries to parse a span of UTF-8 characters into a value.

UInt32)	
TryParse(ReadOnlySpan<Byte>, UInt32)	Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt32)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt32)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, UInt32)	Tries to convert the span representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, UInt32)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, UInt32)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt32)	Tries to convert the string representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<UInt32,UInt32,UInt32>.Addition(UInt32, UInt32)	Adds two values together to compute their sum.
IAdditionOperators<UInt32,UInt32,UInt32>.CheckedAddition(UInt32, UInt32)	Adds two values together to compute their sum.
IAdditiveIdentity<UInt32,UInt32>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<UInt32>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

IBinaryInteger<UInt32>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<UInt32>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UInt32)	
IBinaryInteger<UInt32>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UInt32)	
IBinaryInteger<UInt32>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<UInt32>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<UInt32>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<UInt32,UInt32,UInt32>.BitwiseAnd(UInt32, UInt32)	Computes the bitwise-and of two values.
IBitwiseOperators<UInt32,UInt32,UInt32>.BitwiseOr(UInt32, UInt32)	Computes the bitwise-or of two values.
IBitwiseOperators<UInt32,UInt32,UInt32>.ExclusiveOr(UInt32, UInt32)	Computes the exclusive-or of two values.
IBitwiseOperators<UInt32,UInt32,UInt32>.OnesComplement(UInt32)	Computes the ones-complement representation of a given value.
IComparisonOperators<UInt32,UInt32,Boolean>.GreaterThan(UInt32, UInt32)	Compares two values to determine which is greater.
IComparisonOperators<UInt32,UInt32,Boolean>.GreaterThanOrEqual(UInt32, UInt32)	Compares two values to determine which is greater or equal.
IComparisonOperators<UInt32,UInt32,Boolean>.LessThan(UInt32, UInt32)	Compares two values to determine which is less.
IComparisonOperators<UInt32,UInt32,Boolean>.LessThanOrEqual(UInt32, UInt32)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).

IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<UInt32>.CheckedDecrement(UInt32)	Decrements a value.
IDecrementOperators<UInt32>.Decrement(UInt32)	Decrements a value.
IDivisionOperators<UInt32,UInt32,UInt32>.Division(UInt32, UInt32)	Divides one value by another to compute their quotient.
IEqualityOperators<UInt32,UInt32,Boolean>.Equality(UInt32, UInt32)	Compares two values to determine equality.
IEqualityOperators<UInt32,UInt32,Boolean>.Inequality(UInt32, UInt32)	Compares two values to determine inequality.
IIncrementOperators<UInt32>.CheckedIncrement(UInt32)	Increments a value.

IIncrementOperators<UInt32>.Increment(UInt32)	Increments a value.
IMinMaxValue<UInt32>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<UInt32>.MinValue	Gets the minimum value of the current type.
IModulus Operators<UInt32,UInt32,UInt32>.Modulus(UInt32, UInt32)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<UInt32,UInt32>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<UInt32,UInt32,UInt32>.Checked Multiply(UInt32, UInt32)	Multiplies two values together to compute their product.
IMultiplyOperators<UInt32,UInt32,UInt32>.Multiply(UInt32, UInt32)	Multiplies two values together to compute their product.
INumber<UInt32>.CopySign(UInt32, UInt32)	Copies the sign of a value to the sign of another value.
INumber<UInt32>.MaxNumber(UInt32, UInt32)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<UInt32>.MinNumber(UInt32, UInt32)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<UInt32>.Abs(UInt32)	Computes the absolute of a value.
INumberBase<UInt32>.IsCanonical(UInt32)	Determines if a value is in its canonical representation.
INumberBase<UInt32>.IsComplexNumber(UInt32)	Determines if a value represents a complex number.
INumberBase<UInt32>.IsFinite(UInt32)	Determines if a value is finite.
INumberBase<UInt32>.IsImaginaryNumber(UInt32)	Determines if a value represents a pure imaginary number.
INumberBase<UInt32>.IsInfinity(UInt32)	Determines if a value is infinite.
INumberBase<UInt32>.IsInteger(UInt32)	Determines if a value represents an integral number.
INumberBase<UInt32>.IsNaN(UInt32)	Determines if a value is NaN.

INumberBase<UInt32>.IsNegative(UInt32)	Determines if a value is negative.
INumberBase<UInt32>.IsNegativeInfinity(UInt32)	Determines if a value is negative infinity.
INumberBase<UInt32>.IsNormal(UInt32)	Determines if a value is normal.
INumberBase<UInt32>.IsPositive(UInt32)	Determines if a value is positive.
INumberBase<UInt32>.IsPositiveInfinity(UInt32)	Determines if a value is positive infinity.
INumberBase<UInt32>.IsRealNumber(UInt32)	Determines if a value represents a real number.
INumberBase<UInt32>.IsSubnormal(UInt32)	Determines if a value is subnormal.
INumberBase<UInt32>.IsZero(UInt32)	Determines if a value is zero.
INumberBase<UInt32>.MaxMagnitude(UInt32, UInt32)	Compares two values to compute which is greater.
INumberBase<UInt32>.MaxMagnitudeNumber(UInt32, UInt32)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt32>.MinMagnitude(UInt32, UInt32)	Compares two values to compute which is lesser.
INumberBase<UInt32>.MinMagnitudeNumber(UInt32, UInt32)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt32>.One	Gets the value `1` for the type.
INumberBase<UInt32>.Radix	Gets the radix, or base, for the type.
INumberBase<UInt32>.TryConvertFromChecked<TOther>(TOther, UInt32)	
INumberBase<UInt32>.TryConvertFromSaturating<TOther>(TOther, UInt32)	
INumberBase<UInt32>.TryConvertFromTruncating<TOther>(TOther, UInt32)	
INumberBase<UInt32>.TryConvertToChecked<TOther>(UInt32, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

INumberBase<UInt32>.TryConvertToSaturating<TOther>(UInt32, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<UInt32>.TryConvertToTruncating<TOther>(UInt32, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<UInt32>.Zero	Gets the value `0` for the type.
IShiftOperators<UInt32,Int32,UInt32>.LeftShift(UInt32, Int32)	Shifts a value left by a given amount.
IShiftOperators<UInt32,Int32,UInt32>.RightShift(UInt32, Int32)	Shifts a value right by a given amount.
IShiftOperators<UInt32,Int32,UInt32>.UnsignedRightShift(UInt32, Int32)	Shifts a value right by a given amount.
ISubtractionOperators<UInt32,UInt32,UInt32>.CheckedSubtraction(UInt32, UInt32)	Subtracts two values to compute their difference.
ISubtractionOperators<UInt32,UInt32,UInt32>.Subtraction(UInt32, UInt32)	Subtracts two values to compute their difference.
IUnaryNegationOperators<UInt32,UInt32>.CheckedUnaryNegation(UInt32)	Computes the checked unary negation of a value.
IUnaryNegationOperators<UInt32,UInt32>.UnaryNegation(UInt32)	Computes the unary negation of a value.
IUnaryPlusOperators<UInt32,UInt32>.UnaryPlus(UInt32)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Int32
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of UInt32. This field is constant.

C#

```
public const uint MaxValue = 4294967295;
```

Field Value

Value* = *4294967295

UInt32

Examples

The following example uses the MinValue and MaxValue fields to verify that an Int64 value is within the range of the UInt32 type before it performs a type conversion. This verification prevents an OverflowException at run time.

C#

```
long longValue = long.MaxValue / 2;
uint integerValue;

if (longValue <= uint.MaxValue &&
    longValue >= uint.MinValue)
{
   integerValue = (uint) longValue;
   Console.WriteLine("Converted long integer value to {0:n0}.",
                     integerValue);
}
else
{
   uint rangeLimit;
   string relationship;

   if (longValue > uint.MaxValue)
   {
```

```
        rangeLimit = uint.MaxValue;
        relationship = "greater";
    }
    else
    {
        rangeLimit = uint.MinValue;
        relationship = "less";
    }

    Console.WriteLine("Conversion failure: {0:n0} is {1} than {2:n0}",
                      longValue,
                      relationship,
                      rangeLimit);
}
```

Remarks

The value of this constant is 4,294,967,295; that is, hexadecimal 0xFFFFFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt32.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of UInt32. This field is constant.

C#

```
public const uint MinValue = 0;
```

Field Value

***Value* = 0**

UInt32

Examples

The following example demonstrates how to use the MinValue field to display the smallest possible value of a UInt32 variable.

C#

```
public class Temperature {
    public static uint MinValue {
        get {
            return UInt32.MinValue;
        }
    }

    public static uint MaxValue {
        get {
            return UInt32.MaxValue;
        }
    }

    // The value holder
    protected uint m_value;

    public uint Value {
        get {
```

```
            return m_value;
        }
        set {
            m_value = value;
        }
    }
}
```

Remarks

The value of this constant is 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.Clamp(UInt32, UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static uint Clamp (uint value, uint min, uint max);
```

Parameters

`value` UInt32

The value to clamp.

`min` UInt32

The inclusive minimum to which `value` should clamp.

`max` UInt32

The inclusive maximum to which `value` should clamp.

Returns

UInt32

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or UInt32 and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CompareTo(UInt32)	Compares this instance to a specified 32-bit unsigned integer and returns an indication of their relative values.

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

value Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a UInt32.

Examples

The following code example demonstrates the CompareTo method.

C#

```
public class Temperature : IComparable {
    /// <summary>
    /// IComparable.CompareTo implementation.
    /// </summary>
    public int CompareTo(object obj) {
        if(obj is Temperature) {
            Temperature temp = (Temperature) obj;

            return m_value.CompareTo(temp.m_value);
        }

        throw new ArgumentException("object is not a Temperature");
    }

    // The value holder
    protected uint m_value;

    public uint Value {
        get {
            return m_value;
        }
```

```
        set {
            m_value = value;
        }
    }
}
```

Remarks

Any instance of UInt32, regardless of its value, is considered greater than `null`.

The `value` parameter must be `null` or an instance of UInt32; otherwise, an exception is thrown.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CompareTo(UInt32)

Compares this instance to a specified 32-bit unsigned integer and returns an indication of their relative values.

C#

```
public int CompareTo (uint value);
```

Parameters

`value` UInt32

An unsigned integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";
```

```
    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
```

```
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
```

```
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the UInt32.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static uint CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt32

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static uint CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt32

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.CreateTruncating<TOther> (TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static uint CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

TOther

The type of `value`.

Parameters

value TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt32

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.DivRem(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (uint Quotient, uint Remainder) DivRem (uint left, uint
right);
```

Parameters

`left` UInt32

The value which `right` divides.

`right` UInt32

The value which divides `left`.

Returns

ValueTuple<UInt32,UInt32>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or UInt32.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt32)	Returns a value indicating whether this instance is equal to a specified UInt32.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance.

Returns

Boolean

`true` if `obj` is an instance of UInt32 and equals the value of this instance; otherwise, `false`.

Examples

The following code example demonstrates the Equals method.

C#

```
UInt32 myVariable1 = 20;
UInt32 myVariable2 = 20;

// Display the declaring type.
Console.WriteLine("\nType of 'myVariable1' is '{0}' and"+
     " value is :{1}",myVariable1.GetType(), myVariable1);
Console.WriteLine("Type of 'myVariable2' is '{0}' and"+
     " value is :{1}",myVariable2.GetType(), myVariable2);

// Compare 'myVariable1' instance with 'myVariable2' Object.
if( myVariable1.Equals( myVariable2 ) )
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are equal");
else
   Console.WriteLine( "\nStructures 'myVariable1' and "+
         "'myVariable2' are not equal");
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt32) method overloads. If an implicit conversion between the `obj` argument and a UInt32 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt32) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a UInt32 value. The following example illustrates the difference in behavior between the two method overloads. In the case of Byte and UInt16 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt32) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static uint value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);
```

```
        short short1 = 112;
        Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
        TestObjectForEquality(short1);

        long long1 = 112;
        Console.WriteLine("value = long1: {0,18}", value.Equals(long1));
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,18}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:              True
//       112 (UInt32) = 112 (Byte): False
//
//       value = short1:              False
//       112 (UInt32) = 112 (Int16): False
//
//       value = long1:               False
//       112 (UInt32) = 112 (Int64): False
//
//       value = sbyte1:              False
//       112 (UInt32) = 112 (SByte): False
//
//       value = ushort1:              True
//       112 (UInt32) = 112 (UInt16): False
//
//       value = ulong1:                False
```

```
//       112 (UInt32) = 112 (UInt64): False
//
//       value = dec1:                 False
//       112 (UInt32) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (UInt32) = 112 (Double): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(UInt32)

Returns a value indicating whether this instance is equal to a specified UInt32.

C#

```
public bool Equals (uint obj);
```

Parameters

obj UInt32

A value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than the Equals(Object) method because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt32) method overloads. If an implicit conversion between the `obj` argument and a UInt32 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt32) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a UInt32 value. The following example illustrates the difference in behavior between the two method overloads. In the case of Byte and UInt16 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt32) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static uint value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
      TestObjectForEquality(short1);

      long long1 = 112;
      Console.WriteLine("value = long1: {0,18}", value.Equals(long1));
```

```
        TestObjectForEquality(long1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        ulong ulong1 = 112;
        Console.WriteLine("value = ulong1: {0,18}", value.Equals(ulong1));
        TestObjectForEquality(ulong1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (UInt32) = 112 (Byte): False
//
//       value = short1:             False
//       112 (UInt32) = 112 (Int16): False
//
//       value = long1:              False
//       112 (UInt32) = 112 (Int64): False
//
//       value = sbyte1:             False
//       112 (UInt32) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (UInt32) = 112 (UInt16): False
//
//       value = ulong1:               False
//       112 (UInt32) = 112 (UInt64): False
//
//       value = dec1:                 False
//       112 (UInt32) = 112 (Decimal): False
//
```

```
//       value = dbl1:                False
//       112 (UInt32) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt32.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type UInt32.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, UInt32.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IsEvenInteger(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IsOddInteger(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IsPow2(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.LeadingZeroCount(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

```
C#
public static uint LeadingZeroCount (uint value);
```

Parameters

`value` UInt32

The value whose leading zeroes are to be counted.

Returns

UInt32

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Log2(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static uint Log2 (uint value);
```

Parameters

`value` UInt32

The value whose log2 is to be computed.

Returns

UInt32

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Max(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static uint Max (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Min(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static uint Min (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 32-bit unsigned integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 32-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 32-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Converts the string representation of a number to its 32-bit unsigned integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int64.Parse(String)**

Converts the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent.

C#

```
public static uint Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string representing the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt32

A 32-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example uses the Parse(String, NumberStyles, IFormatProvider) method to convert various string representations of numbers to 32-bit unsigned integer values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] cultureNames= { "en-US", "fr-FR" };
      NumberStyles[] styles= { NumberStyles.Integer,
                               NumberStyles.Integer |
NumberStyles.AllowDecimalPoint };
      string[] values = { "170209", "+170209.0", "+170209,0",
"-103214.00",
                                  "-103214,00", "104561.1", "104561,1" };

      // Parse strings using each culture
      foreach (string cultureName in cultureNames)
      {
         CultureInfo ci = new CultureInfo(cultureName);
         Console.WriteLine("Parsing strings using the {0} culture",
                           ci.DisplayName);
         // Use each style.
```

```
         foreach (NumberStyles style in styles)
         {
            Console.WriteLine("   Style: {0}", style.ToString());
            // Parse each numeric string.
            foreach (string value in values)
            {
               try {
                  Console.WriteLine("      Converted '{0}' to {1}.",
value,
                                    UInt32.Parse(value, style, ci));
               }
               catch (FormatException) {
                  Console.WriteLine("      Unable to parse '{0}'.",
value);
               }
               catch (OverflowException) {
                  Console.WriteLine("      '{0}' is out of range of the
UInt32 type.",
                                    value);
               }
            }
         }
      }
   }
}
// The example displays the following output:
//       Parsing strings using the English (United States) culture
//          Style: Integer
//             Converted '170209' to 170209.
//             Unable to parse '+170209.0'.
//             Unable to parse '+170209,0'.
//             Unable to parse '-103214.00'.
//             Unable to parse '-103214,00'.
//             Unable to parse '104561.1'.
//             Unable to parse '104561,1'.
//          Style: Integer, AllowDecimalPoint
//             Converted '170209' to 170209.
//             Converted '+170209.0' to 170209.
//             Unable to parse '+170209,0'.
//             '-103214.00' is out of range of the UInt32 type.
//             Unable to parse '-103214,00'.
//             '104561.1' is out of range of the UInt32 type.
//             Unable to parse '104561,1'.
//       Parsing strings using the French (France) culture
//          Style: Integer
//             Converted '170209' to 170209.
//             Unable to parse '+170209.0'.
//             Unable to parse '+170209,0'.
//             Unable to parse '-103214.00'.
//             Unable to parse '-103214,00'.
//             Unable to parse '104561.1'.
//             Unable to parse '104561,1'.
//          Style: Integer, AllowDecimalPoint
//             Converted '170209' to 170209.
//             Unable to parse '+170209.0'.
```

```
//              Converted '+170209,0' to 170209.
//              Unable to parse '-103214.00'.
//              '-103214,00' is out of range of the UInt32 type.
//              Unable to parse '104561.1'.
//              '104561,1' is out of range of the UInt32 type.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive or negative sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object that is returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. (The method throws an OverflowException if `s` includes a negative sign and represents a non-zero number.) The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can

Element	Description
	be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0-9 if `style` includes the NumberStyles.AllowExponent flag, or one or more occurrences of the digit 0 if it does not. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, if style does not include the

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
	NumberStyles.AllowExponent flag, *fractional_digits* must consist of only one or more 0 digits; otherwise, an OverflowException is thrown.
NumberStyles.AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign before *digits*.
NumberStyles.AllowTrailingSign	A sign after *digits*.
NumberStyles.AllowParentheses	Parentheses before and after *digits* to indicate a negative value.
NumberStyles.AllowThousands	The group separator (,) element.
NumberStyles.AllowCurrencySymbol	The currency ($) element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. The only other flags that can be combined with it are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If the `s` parameter is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the parse operation to throw an exception.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)

- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent.

```
C#
```

```
public static uint Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt32

A 32-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static uint Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt32

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

Important

This API is not CLS-compliant.

CLS-compliant alternative **System.Int64.Parse(String)**

Converts the string representation of a number in a specified culture-specific format to its 32-bit unsigned integer equivalent.

C#

```
public static uint Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt32

A 32-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct style.

OverflowException

`s` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to a UInt32 value.

C#

```
protected void OkToUInteger_Click(object sender, EventArgs e)
{
    string locale;
    uint number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = UInt32.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

The `s` parameter contains a number of the form:

[*ws*][*sign*]*digits*[*ws*]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign, or a negative sign if `s` represents the value zero.
digits	A sequence of digits ranging from 0 to 9.

The s parameter is interpreted using the NumberStyles.Integer style. In addition to the unsigned integer value's decimal digits, only leading and trailing spaces along with a leading sign is allowed. (If the negative sign is present, `s` must represent a value of zero, or the method throws an OverflowException.) To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()

- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String, NumberStyles)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int64.Parse(String)**

Converts the string representation of a number in a specified style to its 32-bit unsigned integer equivalent.

C#

```
public static uint Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string representing the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of the enumeration values that specify the permitted format of `s`. A typical value to specify is Integer.

Returns

UInt32

A 32-bit unsigned integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in a format compliant with `style`.

OverflowException

`s` represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example tries to parse each element in a string array by using a number of NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
```

```
   public static void Main()
   {
      string[] values= { " 214309 ", "1,064,181", "(0)", "10241+", " +
21499 ",
                         " +21499 ", "122153.00", "1e03ff", "91300.0e-2"
};
      NumberStyles whitespace =  NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite;
      NumberStyles[] styles= { NumberStyles.None, whitespace,
                               NumberStyles.AllowLeadingSign |
NumberStyles.AllowTrailingSign | whitespace,
                               NumberStyles.AllowThousands |
NumberStyles.AllowCurrencySymbol,
                               NumberStyles.AllowExponent |
NumberStyles.AllowDecimalPoint };

      // Attempt to convert each number using each style combination.
      foreach (string value in values)
      {
         Console.WriteLine("Attempting to convert '{0}':", value);
         foreach (NumberStyles style in styles)
         {
            try {
               uint number = UInt32.Parse(value, style);
               Console.WriteLine("   {0}: {1}", style, number);
            }
            catch (FormatException) {
               Console.WriteLine("   {0}: Bad Format", style);
            }
            catch (OverflowException)
            {
               Console.WriteLine("   {0}: Overflow", value);
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    Attempting to convert ' 214309 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: 214309
//       Integer, AllowTrailingSign: 214309
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '1,064,181':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: 1064181
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '(0)':
//       None: Bad Format
```

```
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '10241+':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 10241
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' + 21499 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' +21499 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 21499
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '122153.00':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 122153
//
//    Attempting to convert '1e03ff':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '91300.0e-2':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 913
```

Remarks

The `style` parameter defines the style elements (such as white space, the positive or negative sign symbol, the group separator symbol, or the decimal point symbol) that

are allowed in the `s` parameter for the parse operation to succeed. `style` must be a combination of bit flags from the NumberStyles enumeration. The `style` parameter makes this method overload useful when `s` contains the string representation of a hexadecimal value, when the number system (decimal or hexadecimal) represented by `s` is known only at run time, or when you want to disallow white space or a sign symbol in `s`.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may contain the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits *fractional_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.

Element	Description
exponential_digits	
,	A culture-specific group separator symbol. The current culture's group separator can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, a FormatException is thrown.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the UInt32 type. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

`NumberStyles` value	Elements permitted in `s` in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.

`NumberStyles` value	Elements permitted in `s` in addition to digits
AllowLeadingSign	The *sign* element at the start of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

Unlike the other NumberStyles values, which allow for, but do not require, the presence of particular style elements in `s`, the NumberStyles.AllowHexSpecifier style value means that the individual numeric characters in `s` are always interpreted as hexadecimal characters. Valid hexadecimal characters are 0-9, A-F, and a-f. A prefix, such as "0x", is not allowed. The only other flags that can be combined with the `style` parameter are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

The only other flags that can be combined with the `style` parameter are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static uint Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt32

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

C#

```
public static uint Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt32

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int64.Parse(String)**

Converts the string representation of a number to its 32-bit unsigned integer equivalent.

C#

```
public static uint Parse (string s);
```

Parameters

s String

A string representing the number to convert.

Returns

UInt32

A 32-bit unsigned integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

The `s` parameter is `null`.

FormatException

The `s` parameter is not of the correct format.

OverflowException

The `s` parameter represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue.

Examples

The following example uses the Parse(String) method to parse an array of string values.

C#

```
string[] values = { "+13230", "-0", "1,390,146", "$190,235,421,127",
                    "0xFA1B", "163042", "-10", "2147483648",
                    "14065839182", "16e07", "134985.0", "-12034" };
foreach (string value in values)
{
   try {
      uint number = UInt32.Parse(value);
      Console.WriteLine("{0} --> {1}", value, number);
   }
   catch (FormatException) {
      Console.WriteLine("{0}: Bad Format", value);
   }
   catch (OverflowException) {
      Console.WriteLine("{0}: Overflow", value);
   }
}
// The example displays the following output:
//       +13230 --> 13230
//       -0 --> 0
//       1,390,146: Bad Format
//       $190,235,421,127: Bad Format
//       0xFA1B: Bad Format
//       163042 --> 163042
//       -10: Overflow
//       2147483648 --> 2147483648
//       14065839182: Overflow
//       16e07: Bad Format
//       134985.0: Bad Format
//       -12034: Overflow
```

Remarks

The `s` parameter should be the string representation of a number in the following form.

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits	A sequence of digits ranging from 0 to 9. Any leading zeros are ignored.

> **Note**
>
> The string specified by the `s` parameter is interpreted by using the **NumberStyles.Integer** style. It cannot contain any group separators or decimal separator, and it cannot have a decimal portion.

The `s` parameter is parsed by using the formatting information in a System.Globalization.NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string by using the formatting information of a specific culture, use the Parse(String, IFormatProvider) method.

See also

- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.PopCount(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

```
C#
public static uint PopCount (uint value);
```

Parameters

`value` UInt32

The value whose set bits are to be counted.

Returns

UInt32

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.RotateLeft(UInt32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static uint RotateLeft (uint value, int rotateAmount);
```

Parameters

`value` UInt32

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

UInt32

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.RotateRight(UInt32, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

C#

```
public static uint RotateRight (uint value, int rotateAmount);
```

Parameters

`value` UInt32

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

UInt32

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.Sign(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (uint value);
```

Parameters

`value` UInt32

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString()	Converts the numeric value of this instance to its equivalent string representation.

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance, which consists of a sequence of digits ranging from 0 to 9, without a sign or leading zeros.

Implements

ToString(IFormatProvider)

Examples

The following example formats a 16-bit signed integer value by using several format providers, including one for the invariant culture. The output from the example illustrates that the formatted string returned by the ToString(IFormatProvider) method is the same regardless of the format provider.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define an array of CultureInfo objects.
      CultureInfo[] ci = { new CultureInfo("en-US"),
                           new CultureInfo("fr-FR"),
                           CultureInfo.InvariantCulture };
      uint value = 1870924;
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        GetName(ci[0]), GetName(ci[1]), GetName(ci[2]));
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        value.ToString(ci[0]), value.ToString(ci[1]),
value.ToString(ci[2]));
   }

   private static string GetName(CultureInfo ci)
   {
      if (ci.Equals(CultureInfo.InvariantCulture))
         return "Invariant";
      else
         return ci.Name;
   }
}
```

```
// The example displays the following output:
//              en-US          fr-FR      Invariant
//            1870924        1870924        1870924
```

Remarks

The ToString(IFormatProvider) method formats a UInt32 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific formatting information. However, none of the properties of the NumberFormatInfo are used when formatting with the general numeric format specifier ("G").

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

The `format` parameter is invalid.

Examples

The following example displays a 32-bit unsigned integer value by using each standard format string and some custom format strings.

```
C#

using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
        uint value = 2179608;
        string[] specifiers = { "G", "C", "D3", "E2", "e3", "F",
                                "N", "P", "X", "000000.0", "#.0",
                                "00000000;(0);**Zero**" };

        foreach (string specifier in specifiers)
           Console.WriteLine("{0}: {1}", specifier,
value.ToString(specifier));
    }
}
// The example displays the following output:
//       G: 2179608
//       C: $2,179,608.00
//       D3: 2179608
//       E2: 2.18E+006
//       e3: 2.180e+006
//       F: 2179608.00
//       N: 2,179,608.00
//       P: 217,960,800.00 %
//       X: 214218
//       000000.0: 2179608.0
//       #.0: 2179608.0
//       00000000;(0);**Zero**: 02179608
```

Remarks

The ToString(String) method formats a UInt32 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid Standard Numeric Format Strings, or any combination of Custom Numeric Format Strings. If `format` is equal to String.Empty or is `null`, the return value of the current UInt32 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The format of the returned string is determined by the NumberFormatInfo object for the current culture. Depending on the `format` parameter, this object controls symbols such as the group separator and the decimal point symbol in the output string. To provide formatting information for cultures other than the current culture, call the ToString(String, IFormatProvider) overload.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeroes

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

```
C#
```

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about this instance.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The `format` parameter is invalid.

Examples

The following example displays a 32-bit unsigned integer value by using the standard numeric format specifiers and a number of specific CultureInfo objects.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define cultures whose formatting conventions are to be used.
      CultureInfo[] cultures = { CultureInfo.CreateSpecificCulture("en-
US"),
```

```
                                    CultureInfo.CreateSpecificCulture("fr-
FR"),
                                    CultureInfo.CreateSpecificCulture("es-
ES") };
      string[] specifiers = {"G", "C", "D4", "E2", "F", "N", "P", "X2"};
      uint value = 2222402;

      foreach (string specifier in specifiers)
      {
         foreach (CultureInfo culture in cultures)
            Console.WriteLine("{0,2} format using {1} culture: {2, 18}",
                              specifier, culture.Name,
                              value.ToString(specifier, culture));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//        G format using en-US culture:            2222402
//        G format using fr-FR culture:            2222402
//        G format using es-ES culture:            2222402
//
//        C format using en-US culture:      $2,222,402.00
//        C format using fr-FR culture:     2 222 402,00 €
//        C format using es-ES culture:     2.222.402,00 €
//
//       D4 format using en-US culture:            2222402
//       D4 format using fr-FR culture:            2222402
//       D4 format using es-ES culture:            2222402
//
//       E2 format using en-US culture:          2.22E+006
//       E2 format using fr-FR culture:          2,22E+006
//       E2 format using es-ES culture:          2,22E+006
//
//        F format using en-US culture:         2222402.00
//        F format using fr-FR culture:         2222402,00
//        F format using es-ES culture:         2222402,00
//
//        N format using en-US culture:       2,222,402.00
//        N format using fr-FR culture:       2 222 402,00
//        N format using es-ES culture:       2.222.402,00
//
//        P format using en-US culture:   222,240,200.00 %
//        P format using fr-FR culture:   222 240 200,00 %
//        P format using es-ES culture:   222.240.200,00 %
//
//       X2 format using en-US culture:             21E942
//       X2 format using fr-FR culture:             21E942
//       X2 format using es-ES culture:             21E942
```

Remarks

The ToString(String, IFormatProvider) method formats a UInt32 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be any valid Standard Numeric Format Strings, or any combination of Custom Numeric Format Strings. If `format` is equal to String.Empty or is `null`, the return value of the current UInt32 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string returned by this method. When the ToString(String, IFormatProvider) method is invoked, it calls the `provider` parameter's IFormatProvider.GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information for formatting the current UInt32 value, such as the group separator symbol or the decimal point symbol. There are three ways to use the `provider` parameter to supply formatting information to the ToString(String, IFormatProvider) method:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.

- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the formatting of the returned string is based on the NumberFormatInfo object of the current culture.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a sequence of digits ranging from 0 to 9, without a sign or leading zeroes.

Examples

The following example displays a UInt32 value by using the default ToString() method. It also displays the string representations of the UInt32 value that results from using some standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      uint value = 1632490;
      // Display value using default ToString method.
      Console.WriteLine(value.ToString());
      Console.WriteLine();

      // Define an array of format specifiers.
      string[] formats = { "G", "C", "D", "F", "N", "X" };
      // Display value using the standard format specifiers.
      foreach (string format in formats)
         Console.WriteLine("{0} format specifier: {1,16}",
                           format, value.ToString(format));
   }
}
// The example displays the following output:
//       1632490
//
//       G format specifier:          1632490
//       C format specifier:    $1,632,490.00
//       D format specifier:          1632490
//       F format specifier:       1632490.00
//       N format specifier:     1,632,490.00
//       X format specifier:           18E8EA
```

Remarks

The ToString() method formats a UInt32 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UInt32.TrailingZeroCount(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static uint TrailingZeroCount (uint value);
```

Parameters

`value` UInt32

The value whose trailing zeroes are to be counted.

Returns

UInt32

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current unsigned integer number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current unsigned integer number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the string representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, UInt32)	Tries to convert the span representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt32)	Tries to convert the string representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt32)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, UInt32)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt32)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt32)	Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt32)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(String, NumberStyles, IFormatProvider, UInt32)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt32)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out uint result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt32

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, UInt32)

> Important
>
> This API is not CLS-compliant.

Tries to convert the span representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out uint result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert.

`result` UInt32

When this method returns, contains the 32-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, UInt32)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int64.TryParse(String, Int64)**

Tries to convert the string representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (string? s, out uint result);
```

Parameters

`s` String

A string that represents the number to convert.

`result` UInt32

When this method returns, contains the 32-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParse(String, UInt32) method once for each element in a string array.

```
C#
```

```
string[] numericStrings = { "1293.8", "+1671.7", "28347.",
                            "   33113684  ", "(0)", "-0", "-1",
                            "+1293617", "18-", "119870", "31,024",
                            "  3127094 ", "00700000" };
uint number;
foreach (string numericString in numericStrings)
{
   if (UInt32.TryParse(numericString, out number))
      Console.WriteLine("Converted '{0}' to {1}.", numericString,
number);
   else
      Console.WriteLine("Cannot convert '{0}' to a UInt32.",
numericString);
}
// The example displays the following output:
//       Cannot convert '1293.8' to a UInt32.
//       Cannot convert '+1671.7' to a UInt32.
//       Cannot convert '28347.' to a UInt32.
//       Converted '   33113684  ' to 33113684.
//       Cannot convert '(0)' to a UInt32.
//       Converted '-0' to 0.
//       Cannot convert '-1' to a UInt32.
//       Converted '+1293617' to 1293617.
//       Cannot convert '18-' to a UInt32.
//       Converted '119870' to 119870.
//       Cannot convert '31,024' to a UInt32.
//       Converted '  3127094 ' to 3127094.
//       Converted '0070000' to 70000.
```

Remarks

The TryParse method is like the Parse method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be successfully parsed.

The `s` parameter should be the string representation of a decimal number in the following form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture.
digits	A sequence of decimal digits ranging from 0 to 9.

The `s` parameter is interpreted by using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces with a leading sign are allowed. To explicitly define the style elements with the culture-specific formatting information that can be present in `s`, call the TryParse(String, NumberStyles, IFormatProvider, UInt32) method.

> **Note**
>
> The string specified by the `s` parameter cannot contain any group separators or decimal separator, and it cannot have a fractional portion.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object for the current system culture. For more information, see NumberFormatInfo.CurrentInfo.

This overload interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the TryParse(String, NumberStyles, IFormatProvider, UInt32) overload instead.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt32)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out uint result);
```

Parameters

s ReadOnlySpan<Char>

The span of characters to parse.

provider IFormatProvider

An object that provides culture-specific formatting information about `s`.

result UInt32

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, UInt32)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
uint result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt32

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt32)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
uint result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt32

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, UInt32)

Tries to convert a UTF-8 character span containing the string representation of a number to its 32-bit unsigned integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out uint
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` UInt32

When this method returns, contains the 32-bit unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt32)

Important

This API is not CLS-compliant.

Tries to convert the span representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```C#
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
uint result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt32

When this method returns, contains the 32-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, UInt32)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Int64.TryParse(String, Int64)**

Tries to convert the string representation of a number in a specified style and culture-specific format to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out uint result);
```

Parameters

`s` String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt32

When this method returns, contains the 32-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number that is less than UInt32.MinValue or greater than UInt32.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, UInt32) method with a number of different strings and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "2106034";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-10603";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);
```

```
      numericString = "29103674.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "41792210E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "9112E-01";
      CallTryParse(numericString, styles);

      numericString = "312E01";
      CallTryParse(numericString, styles);

      numericString = "FFC86DA1";
      CallTryParse(numericString, NumberStyles.HexNumber);

      numericString = "0x8F8C";
      CallTryParse(numericString, NumberStyles.HexNumber);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      uint number;
      bool result = UInt32.TryParse(stringToConvert, styles,
                                    CultureInfo.InvariantCulture, out
number);
      if (result)
         Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
      else
         Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
   }
}
// The example displays the following output:
//       Converted '2106034' to 2106034.
//       Attempted conversion of '-10603' failed.
//       Converted '29103674.00' to 29103674.
//       Attempted conversion of '10345.72' failed.
//       Converted '41792210E-01' to 4179221.
//       Attempted conversion of '9112E-01' failed.
//       Converted '312E01' to 3120.
//       Converted 'FFC86DA1' to 4291325345.
//       Attempted conversion of '0x8F8C' failed.
```

Remarks

The TryParse(String, NumberStyles, IFormatProvider, UInt32) method is like the Parse(String, NumberStyles, IFormatProvider) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Items in square brackets ([and]) are optional. Or, if the `style` parameter includes NumberStyles.AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, if the negative sign is present, `s` can only represent the value zero for the parse operation to succeed.
digits	A sequence of digits from 0 through 9.

Element	Description
,	A culture-specific group separator. The group separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
None	Decimal digits only.
AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits, or the method returns `false`.

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*. If `s` represents a number in exponential notation, it cannot have a non-zero, fractional component.
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element before *digits*.
AllowTrailingSign	The *sign* element after *digits*.
AllowParentheses	The *sign* element in the form of parentheses enclosing a zero numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The *ws*, *sign*, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, HexNumber, that includes both white-space flags.)

> **Note**
>
> If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. The `provider` parameter can be any one of the following:

- A CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- A NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- A custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String)
- ToString()
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSByte(IFormatProvider).

C#

```
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSByte(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this UInt32 value.

provider IFormatProvider

An IFormatProvider implementation that supplies culture-specific information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToUInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt16(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, unchanged.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, converted to a UInt64.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt32 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt64(UInt32) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.IAdditionOperators<UInt32,UInt32,UInt32>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static uint IAdditionOperators<uint,uint,uint>.operator + (uint left, uint right);
```

Parameters

`left` UInt32

The value to which `right` is added.

`right` UInt32

The value that's added to `left`.

Returns

UInt32

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IAdditionOperators<UInt32,UInt32,UInt32>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static uint IAdditionOperators<uint,uint,uint>.op_CheckedAddition (uint
left, uint right);
```

Parameters

`left` UInt32

The value to which `right` is added.

`right` UInt32

The value that's added to `left`.

Returns

UInt32

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IAdditiveIdentity<UInt32,UInt32>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static uint
System.Numerics.IAdditiveIdentity<System.UInt32,System.UInt32>.AdditiveIdentity { get; }
```

Property Value

UInt32

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt32.IBinaryInteger<UInt32>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

```
C#

int IBinaryInteger<uint>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt32.IBinaryInteger<UInt32>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<uint>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt32.IBinaryInteger<UInt32>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<uint>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out uint value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt32.IBinaryInteger<UInt32>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<uint>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out uint value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt32.IBinaryInteger<UInt32>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<uint>.TryWriteBigEndian (Span<byte> destination, out int
bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBinaryInteger<UInt32>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<uint>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBinaryNumber<UInt32>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static uint System.Numerics.IBinaryNumber<System.UInt32>.AllBitsSet { get; }
```

Property Value

UInt32

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBitwiseOperators<UInt32,UInt32,UInt32>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static uint IBitwiseOperators<uint,uint,uint>.operator & (uint left, uint right);
```

Parameters

`left` UInt32

The value to `and` with `right`.

`right` UInt32

The value to `and` with `left`.

Returns

UInt32

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBitwiseOperators<UInt32,UInt32,UInt32>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static uint IBitwiseOperators<uint,uint,uint>.operator | (uint left, uint right);
```

Parameters

`left` UInt32

The value to `or` with `right`.

`right` UInt32

The value to `or` with `left`.

Returns

UInt32

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBitwiseOperators<UInt32,UInt32,UInt32>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static uint IBitwiseOperators<uint,uint,uint>.operator ^ (uint left, uint right);
```

Parameters

`left` UInt32

The value to xor with `right`.

`right` UInt32

The value to xor with `left`.

Returns

UInt32

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IBitwiseOperators<UInt32,UInt32,UInt32>.OnesComplementOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static uint IBitwiseOperators<uint,uint,uint>.operator ~ (uint value);
```

Parameters

`value` UInt32

The value for which to compute the ones-complement.

Returns

UInt32

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IComparisonOperators<UInt32,UInt32,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<uint,uint,bool>.operator > (uint left, uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IComparisonOperators<UInt32,UInt32,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<uint,uint,bool>.operator >= (uint left,
uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IComparisonOperators<UInt32,UInt32,Boolean>.LessThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<uint,uint,bool>.operator < (uint left, uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IComparisonOperators<UInt32,UInt32,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<uint,uint,bool>.operator <= (uint left,
uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IDecrementOperators<UInt32>.CheckedDecrement(UInt32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#

static uint IDecrementOperators<uint>.op_CheckedDecrement (uint value);
```

Parameters

`value` UInt32

The value to decrement.

Returns

UInt32

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IDecrementOperators<UInt32>.Decrement(UInt32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static uint IDecrementOperators<uint>.operator -- (uint value);
```

Parameters

`value` UInt32

The value to decrement.

Returns

UInt32

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IDivisionOperators<UInt32,UInt32,UInt32>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static uint IDivisionOperators<uint,uint,uint>.operator / (uint left, uint right);
```

Parameters

`left` UInt32

The value that `right` divides.

`right` UInt32

The value that divides `left`.

Returns

UInt32

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IEqualityOperators<UInt32,UInt32,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<uint,uint,bool>.operator == (uint left, uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IEqualityOperators<UInt32,UInt32,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<uint,uint,bool>.operator != (uint left, uint right);
```

Parameters

`left` UInt32

The value to compare with `right`.

`right` UInt32

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IIncrementOperators<UInt32>.CheckedIncrement(UInt32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#

static uint IIncrementOperators<uint>.op_CheckedIncrement (uint value);
```

Parameters

`value` UInt32

The value to increment.

Returns

UInt32

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IIncrementOperators<UInt32>.Increment(UInt32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static uint IIncrementOperators<uint>.operator ++ (uint value);
```

Parameters

`value` UInt32

The value to increment.

Returns

UInt32

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IMinMaxValue<UInt32>.Max Value Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static uint System.Numerics.IMinMaxValue<System.UInt32>.MaxValue { get; }
```

Property Value

UInt32

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IMinMaxValue<UInt32>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static uint System.Numerics.IMinMaxValue<System.UInt32>.MinValue { get; }
```

Property Value

UInt32

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IModulusOperators<UInt32,UInt32,UInt32>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static uint IModulusOperators<uint,uint,uint>.operator % (uint left, uint right);
```

Parameters

`left` UInt32

The value that `right` divides.

`right` UInt32

The value that divides `left`.

Returns

UInt32

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IMultiplicativeIdentity<UInt32,UInt32>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static uint
System.Numerics.IMultiplicativeIdentity<System.UInt32,System.UInt32>.MultiplicativeIdentity { get; }
```

Property Value

UInt32

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt32.IMultiplyOperators<UInt32,UInt32,UInt32>.CheckedMultiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static uint IMultiplyOperators<uint,uint,uint>.op_CheckedMultiply (uint
left, uint right);
```

Parameters

`left` UInt32

The value that `right` multiplies.

`right` UInt32

The value that multiplies `left`.

Returns

UInt32

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IMultiplyOperators<UInt32,UInt32,UInt32>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static uint IMultiplyOperators<uint,uint,uint>.operator * (uint left, uint right);
```

Parameters

`left` UInt32

The value that `right` multiplies.

`right` UInt32

The value that multiplies `left`.

Returns

UInt32

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumber<UInt32>.CopySign(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static uint INumber<uint>.CopySign (uint value, uint sign);
```

Parameters

`value` UInt32

The value whose magnitude is used in the result.

`sign` UInt32

The value whose sign is used in the result.

Returns

UInt32

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumber<UInt32>.MaxNumber(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static uint INumber<uint>.MaxNumber (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumber<UInt32>.MinNumber(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

C#

```
static uint INumber<uint>.MinNumber (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.Abs(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static uint INumberBase<uint>.Abs (uint value);
```

Parameters

`value` UInt32

The value for which to get its absolute.

Returns

UInt32

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsCanonical(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<uint>.IsCanonical (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsComplexNumber(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

```
C#
static bool INumberBase<uint>.IsComplexNumber (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsFinite(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<uint>.IsFinite (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsImaginaryNumber(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<uint>.IsImaginaryNumber (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsInfinity(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<uint>.IsInfinity (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsInteger(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<uint>.IsInteger (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsNaN(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<uint>.IsNaN (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsNegative(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<uint>.IsNegative (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsNegativeInfinity(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<uint>.IsNegativeInfinity (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsNormal(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<uint>.IsNormal (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsPositive(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<uint>.IsPositive (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsPositiveInfinity(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<uint>.IsPositiveInfinity (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsRealNumber(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<uint>.IsRealNumber (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsSubnormal(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<uint>.IsSubnormal (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.IsZero(UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<uint>.IsZero (uint value);
```

Parameters

`value` UInt32

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.MaxMagnitude(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static uint INumberBase<uint>.MaxMagnitude (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.MaxMagnitudeNumber(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static uint INumberBase<uint>.MaxMagnitudeNumber (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.MinMagnitude(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
static uint INumberBase<uint>.MinMagnitude (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.MinMagnitudeNumber(UInt32, UInt32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static uint INumberBase<uint>.MinMagnitudeNumber (uint x, uint y);
```

Parameters

x UInt32

The value to compare with `y`.

y UInt32

The value to compare with `x`.

Returns

UInt32

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

```
C#
static uint System.Numerics.INumberBase<System.UInt32>.One { get; }
```

Property Value

UInt32

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt32.INumberBase<UInt32>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.UInt32>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<uint>.TryConvertFromChecked<TOther> (TOther value,
out uint result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<uint>.TryConvertFromSaturating<TOther> (TOther
value, out uint result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<uint>.TryConvertFromTruncating<TOther> (TOther
value, out uint result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt32

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<uint>.TryConvertToChecked<TOther> (uint value, out
TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<uint>.TryConvertToSaturating<TOther> (uint value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<uint>.TryConvertToTruncating<TOther> (uint value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt32

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.INumberBase<UInt32>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

```
C#
static uint System.Numerics.INumberBase<System.UInt32>.Zero { get; }
```

Property Value

UInt32

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt32.IShift Operators<UInt32,Int32,UInt32>.Left Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static uint IShiftOperators<uint,int,uint>.operator << (uint value, int
shiftAmount);
```

Parameters

`value` UInt32

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

UInt32

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IShift Operators<UInt32,Int32,UInt32>.Right Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static uint IShiftOperators<uint,int,uint>.operator >> (uint value, int
shiftAmount);
```

Parameters

`value` UInt32

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt32

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IShift Operators<UInt32,Int32,UInt32>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static uint IShiftOperators<uint,int,uint>.op_UnsignedRightShift (uint
value, int shiftAmount);
```

Parameters

`value` UInt32

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt32

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.ISubtractionOperators<UInt32,UInt32,UInt32>.CheckedSubtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static uint ISubtractionOperators<uint,uint,uint>.op_CheckedSubtraction
(uint left, uint right);
```

Parameters

`left` UInt32

The value from which `right` is subtracted.

`right` UInt32

The value that's subtracted from `left`.

Returns

UInt32

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.ISubtractionOperators<UInt32,UInt32,UInt32>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static uint ISubtractionOperators<uint,uint,uint>.operator - (uint left,
uint right);
```

Parameters

`left` UInt32

The value from which `right` is subtracted.

`right` UInt32

The value that's subtracted from `left`.

Returns

UInt32

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IUnaryNegationOperators<UInt32,UInt32>.CheckedUnaryNegation Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static uint IUnaryNegationOperators<uint,uint>.op_CheckedUnaryNegation (uint
value);
```

Parameters

`value` UInt32

The value for which to compute the unary negation.

Returns

UInt32

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IUnaryNegationOperators<UInt32,UInt32>.UnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static uint IUnaryNegationOperators<uint,uint>.operator - (uint value);
```

Parameters

`value` UInt32

The value for which to compute the unary negation.

Returns

UInt32

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt32.IUnaryPlusOperators<UInt32,UInt32>.UnaryPlus(UInt32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static uint IUnaryPlusOperators<uint,uint>.operator + (uint value);
```

Parameters

`value` UInt32

The value for which to compute the unary plus.

Returns

UInt32

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64 Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents a 64-bit unsigned integer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UInt64 : IComparable<ulong>, IConvertible,
IEquatable<ulong>, IParsable<ulong>, ISpanParsable<ulong>,
IUtf8SpanParsable<ulong>,
System.Numerics.IAdditionOperators<ulong,ulong,ulong>,
System.Numerics.IAdditiveIdentity<ulong,ulong>,
System.Numerics.IBinaryInteger<ulong>, System.Numerics.IBinaryNumber<ulong>,
System.Numerics.IBitwiseOperators<ulong,ulong,ulong>,
System.Numerics.IComparisonOperators<ulong,ulong,bool>,
System.Numerics.IDecrementOperators<ulong>,
System.Numerics.IDivisionOperators<ulong,ulong,ulong>,
System.Numerics.IEqualityOperators<ulong,ulong,bool>,
System.Numerics.IIncrementOperators<ulong>,
System.Numerics.IMinMaxValue<ulong>,
System.Numerics.IModulusOperators<ulong,ulong,ulong>,
System.Numerics.IMultiplicativeIdentity<ulong,ulong>,
System.Numerics.IMultiplyOperators<ulong,ulong,ulong>,
System.Numerics.INumber<ulong>, System.Numerics.INumberBase<ulong>,
System.Numerics.IShiftOperators<ulong,int,ulong>,
System.Numerics.ISubtractionOperators<ulong,ulong,ulong>,
System.Numerics.IUnaryNegationOperators<ulong,ulong>,
System.Numerics.IUnaryPlusOperators<ulong,ulong>,
System.Numerics.IUnsignedNumber<ulong>
```

Inheritance Object → ValueType → UInt64

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UInt64> , IConvertible , IEquatable<UInt64> , IFormattable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<UInt64> , IParsable<TSelf> , ISpanParsable<UInt64> , ISpanParsable<TSelf> , IAdditionOperators<UInt64,UInt64,UInt64> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UInt64,UInt64> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UInt64> , IBinaryNumber<UInt64> , IBinaryNumber<TSelf> , IBitwiseOperators<UInt64,UInt64,UInt64> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UInt64,UInt64,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UInt64> , IDecrementOperators<TSelf> , IDivisionOperators<UInt64,UInt64,UInt64> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UInt64,UInt64,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UInt64> , IIncrementOperators<TSelf> , IMinMaxValue<UInt64> , IModulusOperators<UInt64,UInt64,UInt64> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UInt64,UInt64> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UInt64,UInt64,UInt64> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UInt64> , INumber<TSelf> , INumberBase<UInt64> , INumberBase<TSelf> , IShiftOperators<UInt64,Int32,UInt64> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UInt64,UInt64,UInt64> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UInt64,UInt64> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<UInt64,UInt64> , IUnaryPlusOperators<TSelf,TSelf> ,

IUnsignedNumber<UInt64> , IUtf8SpanFormattable ,

IUtf8SpanParsable<UInt64> , IUtf8SpanParsable<TSelf>

Remarks

The UInt64 value type represents unsigned integers with values ranging from 0 to 18,446,744,073,709,551,615.

> **Important**
>
> The **UInt64** type is not CLS-compliant. The CLS-compliant alternative type is **Decimal**. **Int64** can be used instead to replace a **UInt64** value that ranges from zero to **MaxValue**. For more information about CLS compliance, see **Language Independence and Language-Independent Components**.

UInt64 provides methods to compare instances of this type, convert the value of an instance to its string representation, and convert the string representation of a number to an instance of this type.

Fields

Expand table

MaxValue	Represents the largest possible value of UInt64. This field is constant.
MinValue	Represents the smallest possible value of UInt64. This field is constant.

Methods

Expand table

Clamp(UInt64, UInt64, UInt64)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CompareTo(UInt64)	Compares this instance to a specified 64-bit unsigned integer and returns an indication of their relative values.

CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(UInt64, UInt64)	Computes the quotient and remainder of two values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt64)	Returns a value indicating whether this instance is equal to a specified UInt64 value.
GetHashCode()	Returns the hash code for this instance.
GetTypeCode()	Returns the TypeCode for value type UInt64.
IsEvenInteger(UInt64)	Determines if a value represents an even integral number.
IsOddInteger(UInt64)	Determines if a value represents an odd integral number.
IsPow2(UInt64)	Determines if a value is a power of two.
LeadingZeroCount(UInt64)	Computes the number of leading zeros in a value.
Log2(UInt64)	Computes the log2 of a value.
Max(UInt64, UInt64)	Compares two values to compute which is greater.
Min(UInt64, UInt64)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.
Parse(String)	Converts the string representation of a number to its 64-bit

	unsigned integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 64-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 64-bit unsigned integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.
PopCount(UInt64)	Computes the number of bits that are set in a value.
RotateLeft(UInt64, Int32)	Rotates a value left by a given amount.
RotateRight(UInt64, Int32)	Rotates a value right by a given amount.
Sign(UInt64)	Computes the sign of a value.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
TrailingZeroCount(UInt64)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current unsigned long number instance into the provided span of characters.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt64)	Tries to parse a span of UTF-8 characters into a value.

TryParse(ReadOnlySpan<Byte>, UInt64)	Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt64)	Tries to parse a span of characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt64)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, UInt64)	Tries to convert the span representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, IFormatProvider, UInt64)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, UInt64)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt64)	Tries to convert the string representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

Expand table

IAdditionOperators<UInt64,UInt64,UInt64>.Addition(UInt64, UInt64)	Adds two values together to compute their sum.
IAdditionOperators<UInt64,UInt64,UInt64>.CheckedAddition(UInt64, UInt64)	Adds two values together to compute their sum.
IAdditiveIdentity<UInt64,UInt64>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<UInt64>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<UInt64>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement

	representation of the current value.
IBinaryInteger<UInt64>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UInt64)	
IBinaryInteger<UInt64>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UInt64)	
IBinaryInteger<UInt64>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<UInt64>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<UInt64>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<UInt64,UInt64,UInt64>.BitwiseAnd(UInt64, UInt64)	Computes the bitwise-and of two values.
IBitwiseOperators<UInt64,UInt64,UInt64>.BitwiseOr(UInt64, UInt64)	Computes the bitwise-or of two values.
IBitwiseOperators<UInt64,UInt64,UInt64>.ExclusiveOr(UInt64, UInt64)	Computes the exclusive-or of two values.
IBitwiseOperators<UInt64,UInt64,UInt64>.OnesComplement(UInt64)	Computes the ones-complement representation of a given value.
IComparisonOperators<UInt64,UInt64,Boolean>.GreaterThan(UInt64, UInt64)	Compares two values to determine which is greater.
IComparisonOperators<UInt64,UInt64,Boolean>.GreaterThanOrEqual(UInt64, UInt64)	Compares two values to determine which is greater or equal.
IComparisonOperators<UInt64,UInt64,Boolean>.LessThan(UInt64, UInt64)	Compares two values to determine which is less.
IComparisonOperators<UInt64,UInt64,Boolean>.LessThanOrEqual(UInt64, UInt64)	Compares two values to determine which is less or equal.
IConvertible.ToBoolean(IFormatProvider)	For a description of this member, see ToBoolean(IFormatProvider).
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	This conversion is not supported. Attempting to use this method throws an InvalidCastException.

IConvertible.ToDecimal(IFormatProvider)	For a description of this member, see ToDecimal(IFormatProvider).
IConvertible.ToDouble(IFormatProvider)	For a description of this member, see ToDouble(IFormatProvider).
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).
IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	For a description of this member, see ToSingle(IFormatProvider).
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<UInt64>.CheckedDecrement(UInt64)	Decrements a value.
IDecrementOperators<UInt64>.Decrement(UInt64)	Decrements a value.
IDivisionOperators<UInt64,UInt64,UInt64>.Division(UInt64, UInt64)	Divides one value by another to compute their quotient.
IEquality Operators<UInt64,UInt64,Boolean>.Equality(UInt64, UInt64)	Compares two values to determine equality.
IEquality Operators<UInt64,UInt64,Boolean>.Inequality(UInt64, UInt64)	Compares two values to determine inequality.
IIncrementOperators<UInt64>.CheckedIncrement(UInt64)	Increments a value.
IIncrementOperators<UInt64>.Increment(UInt64)	Increments a value.

IMinMaxValue<UInt64>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<UInt64>.MinValue	Gets the minimum value of the current type.
IModulus Operators<UInt64,UInt64,UInt64>.Modulus(UInt64, UInt64)	Divides two values together to compute their modulus or remainder.
IMultiplicativeIdentity<UInt64,UInt64>.Multiplicative Identity	Gets the multiplicative identity of the current type.
IMultiplyOperators<UInt64,UInt64,UInt64>.Checked Multiply(UInt64, UInt64)	Multiplies two values together to compute their product.
IMultiplyOperators<UInt64,UInt64,UInt64>.Multiply(UInt64, UInt64)	Multiplies two values together to compute their product.
INumber<UInt64>.CopySign(UInt64, UInt64)	Copies the sign of a value to the sign of another value.
INumber<UInt64>.MaxNumber(UInt64, UInt64)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<UInt64>.MinNumber(UInt64, UInt64)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<UInt64>.Abs(UInt64)	Computes the absolute of a value.
INumberBase<UInt64>.IsCanonical(UInt64)	Determines if a value is in its canonical representation.
INumberBase<UInt64>.IsComplexNumber(UInt64)	Determines if a value represents a complex number.
INumberBase<UInt64>.IsFinite(UInt64)	Determines if a value is finite.
INumberBase<UInt64>.IsImaginaryNumber(UInt64)	Determines if a value represents a pure imaginary number.
INumberBase<UInt64>.IsInfinity(UInt64)	Determines if a value is infinite.
INumberBase<UInt64>.IsInteger(UInt64)	Determines if a value represents an integral number.
INumberBase<UInt64>.IsNaN(UInt64)	Determines if a value is NaN.
INumberBase<UInt64>.IsNegative(UInt64)	Determines if a value is negative.

INumberBase<UInt64>.IsNegativeInfinity(UInt64)	Determines if a value is negative infinity.
INumberBase<UInt64>.IsNormal(UInt64)	Determines if a value is normal.
INumberBase<UInt64>.IsPositive(UInt64)	Determines if a value is positive.
INumberBase<UInt64>.IsPositiveInfinity(UInt64)	Determines if a value is positive infinity.
INumberBase<UInt64>.IsRealNumber(UInt64)	Determines if a value represents a real number.
INumberBase<UInt64>.IsSubnormal(UInt64)	Determines if a value is subnormal.
INumberBase<UInt64>.IsZero(UInt64)	Determines if a value is zero.
INumberBase<UInt64>.MaxMagnitude(UInt64, UInt64)	Compares two values to compute which is greater.
INumberBase<UInt64>.MaxMagnitudeNumber(UInt64, UInt64)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt64>.MinMagnitude(UInt64, UInt64)	Compares two values to compute which is lesser.
INumberBase<UInt64>.MinMagnitudeNumber(UInt64, UInt64)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<UInt64>.One	Gets the value `1` for the type.
INumberBase<UInt64>.Radix	Gets the radix, or base, for the type.
INumberBase<UInt64>.TryConvertFromChecked<TOther>(TOther, UInt64)	
INumberBase<UInt64>.TryConvertFromSaturating<TOther>(TOther, UInt64)	
INumberBase<UInt64>.TryConvertFromTruncating<TOther>(TOther, UInt64)	
INumberBase<UInt64>.TryConvertToChecked<TOther>(UInt64, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<UInt64>.TryConvertToSaturating<TOther>(UInt64, TOther)	Tries to convert an instance of the the current type to another type,

	saturating any values that fall outside the representable range of the current type.
INumberBase<UInt64>.TryConvertToTruncating<TOther>(UInt64, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<UInt64>.Zero	Gets the value `0` for the type.
IShiftOperators<UInt64,Int32,UInt64>.LeftShift(UInt64, Int32)	Shifts a value left by a given amount.
IShiftOperators<UInt64,Int32,UInt64>.RightShift(UInt64, Int32)	Shifts a value right by a given amount.
IShiftOperators<UInt64,Int32,UInt64>.UnsignedRightShift(UInt64, Int32)	Shifts a value right by a given amount.
ISubtractionOperators<UInt64,UInt64,UInt64>.CheckedSubtraction(UInt64, UInt64)	Subtracts two values to compute their difference.
ISubtractionOperators<UInt64,UInt64,UInt64>.Subtraction(UInt64, UInt64)	Subtracts two values to compute their difference.
IUnaryNegationOperators<UInt64,UInt64>.CheckedUnaryNegation(UInt64)	Computes the checked unary negation of a value.
IUnaryNegationOperators<UInt64,UInt64>.UnaryNegation(UInt64)	Computes the unary negation of a value.
IUnaryPlusOperators<UInt64,UInt64>.UnaryPlus(UInt64)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- Int64
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.MaxValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of UInt64. This field is constant.

C#

```
public const ulong MaxValue = 18446744073709551615;
```

Field Value

Value* = *18446744073709551615

UInt64

Examples

The following example uses the MinValue and MaxValue fields to verify that a Double value is within the range of the UInt64 type before it performs a type conversion. This prevents an OverflowException at run time.

C#

```
double decimalValue = -1.5;
ulong integerValue;

// Discard fractional portion of Double value
double decimalInteger = Math.Floor(decimalValue);

if (decimalInteger <= ulong.MaxValue &&
    decimalInteger >= ulong.MinValue)
{
   integerValue = (ulong) decimalValue;
   Console.WriteLine("Converted {0} to {1}.", decimalValue, integerValue);
}
else
{
   ulong rangeLimit;
   string relationship;
```

```
    if (decimalInteger > ulong.MaxValue)
    {
       rangeLimit = ulong.MaxValue;
       relationship = "greater";
    }
    else
    {
       rangeLimit = ulong.MinValue;
       relationship = "less";
    }

    Console.WriteLine("Conversion failure: {0} is {1} than {2}.",
                      decimalInteger,
                      relationship,
                      rangeLimit);
 }
```

Remarks

The value of this constant is 18,446,744,073,709,551,615; that is, hexadecimal 0xFFFFFFFFFFFFFFFF.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt64.MinValue Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of UInt64. This field is constant.

C#

```
public const ulong MinValue = 0;
```

Field Value

Value* = *0

UInt64

Examples

The following example uses the MinValue and MaxValue fields to verify that a Double value is within the range of the UInt64 type before it performs a type conversion. This prevents an OverflowException at run time.

C#

```
double decimalValue = -1.5;
ulong integerValue;

// Discard fractional portion of Double value
double decimalInteger = Math.Floor(decimalValue);

if (decimalInteger <= ulong.MaxValue &&
    decimalInteger >= ulong.MinValue)
{
   integerValue = (ulong) decimalValue;
   Console.WriteLine("Converted {0} to {1}.", decimalValue, integerValue);
}
else
{
   ulong rangeLimit;
   string relationship;
```

```
    if (decimalInteger > ulong.MaxValue)
    {
       rangeLimit = ulong.MaxValue;
       relationship = "greater";
    }
    else
    {
       rangeLimit = ulong.MinValue;
       relationship = "less";
    }

    Console.WriteLine("Conversion failure: {0} is {1} than {2}.",
                      decimalInteger,
                      relationship,
                      rangeLimit);
 }
```

Remarks

The value of this constant is 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MaxValue

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UInt64.Clamp(UInt64, UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

```
C#
public static ulong Clamp (ulong value, ulong min, ulong max);
```

Parameters

`value` UInt64

The value to clamp.

`min` UInt64

The inclusive minimum to which `value` should clamp.

`max` UInt64

The inclusive maximum to which `value` should clamp.

Returns

UInt64

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this instance to a specified object or UInt64 and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Object)	Compares this instance to a specified object and returns an indication of their relative values.
CompareTo(UInt64)	Compares this instance to a specified 64-bit unsigned integer and returns an indication of their relative values.

CompareTo(Object)

Compares this instance to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`, or `value` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`value` is not a UInt64.

Examples

The following code example demonstrates the CompareTo method.

C#

```
public class Temperature : IComparable {
    /// <summary>
    /// IComparable.CompareTo implementation.
    /// </summary>
    public int CompareTo(object obj) {
        if(obj is Temperature) {
            Temperature temp = (Temperature) obj;

            return m_value.CompareTo(temp.m_value);
        }

        throw new ArgumentException("object is not a Temperature");
    }

    // The value holder
    protected ulong m_value;

    public ulong Value {
        get {
            return m_value;
        }
```

```
        set {
            m_value = value;
        }
    }
}
```

Remarks

Any instance of UInt64, regardless of its value, is considered greater than `null`.

`value` must be `null` or an instance of UInt64; otherwise, an exception is thrown.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

CompareTo(UInt64)

Compares this instance to a specified 64-bit unsigned integer and returns an indication of their relative values.

C#

```
public int CompareTo (ulong value);
```

Parameters

`value` UInt64

An unsigned integer to compare.

Returns

Int32

A signed number indicating the relative values of this instance and `value`.

Expand table

Return Value	Description
Less than zero	This instance is less than `value`.
Zero	This instance is equal to `value`.
Greater than zero	This instance is greater than `value`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the CompareTo method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
```

```
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
```

```
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
```

```
of the
CompareTo method for several base types:

Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface and performs slightly better than the UInt64.CompareTo method because it does not have to convert the `value` parameter to an object.

Depending on your programming language, it might be possible to code a CompareTo method where the parameter type has fewer bits (is narrower) than the instance type. This is possible because some programming languages perform an implicit widening conversion that represents the parameter as a type with as many bits as the instance.

For example, suppose the instance type is Int32 and the parameter type is Byte. The Microsoft C# compiler generates instructions to represent the value of the parameter as an Int32 object, then generates a Int32.CompareTo method that compares the values of the Int32 instance and the Int32 parameter representation.

Consult your programming language's documentation to determine whether its compiler performs implicit widening conversions on numeric types.

See also

- Equals(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
public static ulong CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt64

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

C#

```
public static ulong CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt64

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

C#

```
public static ulong CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UInt64

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.DivRem(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (ulong Quotient, ulong Remainder) DivRem (ulong left, ulong
right);
```

Parameters

`left` UInt64

The value which `right` divides.

`right` UInt64

The value which divides `left`.

Returns

ValueTuple<UInt64,UInt64>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether this instance is equal to a specified object or UInt64.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UInt64)	Returns a value indicating whether this instance is equal to a specified UInt64 value.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare to this instance.

Returns

Boolean

`true` if `obj` is an instance of UInt64 and equals the value of this instance; otherwise, `false`.

Examples

The following example demonstrates the Equals method.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      object[] values = { (short) 10, (short) 20, 10, 20,
                          10L, 20L, 10D, 20D, (ushort) 10,
                          (ushort) 20, 10U, 20U,
                          10ul, 20ul };
      UInt64 baseValue = 20;
      String baseType = baseValue.GetType().Name;

      foreach (var value in values)
         Console.WriteLine("{0} ({1}) = {2} ({3}): {4}",
                           baseValue, baseType,
                           value, value.GetType().Name,
                           baseValue.Equals(value));
   }
}
// The example displays the following output:
//       20 (UInt64) = 10 (Int16): False
//       20 (UInt64) = 20 (Int16): False
//       20 (UInt64) = 10 (Int32): False
//       20 (UInt64) = 20 (Int32): False
//       20 (UInt64) = 10 (Int64): False
//       20 (UInt64) = 20 (Int64): False
//       20 (UInt64) = 10 (Double): False
//       20 (UInt64) = 20 (Double): False
//       20 (UInt64) = 10 (UInt16): False
//       20 (UInt64) = 20 (UInt16): False
//       20 (UInt64) = 10 (UInt32): False
//       20 (UInt64) = 20 (UInt32): False
//       20 (UInt64) = 10 (UInt64): False
//       20 (UInt64) = 20 (UInt64): True
```

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt64) method overloads. If an implicit conversion between the `obj` argument and a UInt64 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt64) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj`

argument is not a UInt64 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, UInt16, and UInt32 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt64) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static ulong value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
      TestObjectForEquality(byte1);

      short short1 = 112;
      Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
      TestObjectForEquality(short1);

      int int1 = 112;
      Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
      TestObjectForEquality(int1);

      sbyte sbyte1 = 112;
      Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
      TestObjectForEquality(sbyte1);

      ushort ushort1 = 112;
      Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
      TestObjectForEquality(ushort1);

      uint uint1 = 112;
      Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
      TestObjectForEquality(uint1);

      decimal dec1 = 112m;
      Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
      TestObjectForEquality(dec1);

      double dbl1 = 112;
      Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
      TestObjectForEquality(dbl1);
   }

   private static void TestObjectForEquality(Object obj)
```

```
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (UInt64) = 112 (Byte): False
//
//       value = short1:             False
//       112 (UInt64) = 112 (Int16): False
//
//       value = int1:               False
//       112 (UInt64) = 112 (Int32): False
//
//       value = sbyte1:             False
//       112 (UInt64) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (UInt64) = 112 (UInt16): False
//
//       value = uint1:               True
//       112 (UInt64) = 112 (UInt32): False
//
//       value = dec1:                 False
//       112 (UInt64) = 112 (Decimal): False
//
//       value = dbl1:                False
//       112 (UInt64) = 112 (Double): False
```

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Equals(UInt64)

Returns a value indicating whether this instance is equal to a specified UInt64 value.

C#

```
public bool Equals (ulong obj);
```

Parameters

obj UInt64

A UInt64 value to compare to this instance.

Returns

Boolean

`true` if `obj` has the same value as this instance; otherwise, `false`.

Implements

Equals(T)

Examples

The following example demonstrates the Equals method.

C#

```
using System;

class Example
{
     public static void Main()
     {
         UInt64 value1 = 50;
         UInt64 value2 = 50;

         // Display the values.
        Console.WriteLine("value1:   Type: {0}   Value: {1}",
```

```
                                value1.GetType().Name, value1);
        Console.WriteLine("value2:   Type: {0}   Value: {1}",
                                value2.GetType().Name, value2);

        // Compare the two values.
        Console.WriteLine("value1 and value2 are equal: {0}",
                                value1.Equals(value2));
    }
}
// The example displays the following output:
//       value1:   Type: UInt64   Value: 50
//       value2:   Type: UInt64   Value: 50
//       value1 and value2 are equal: True
```

Remarks

This method implements the System.IEquatable<T> interface, and performs slightly better than Equals because it does not have to convert the `obj` parameter to an object.

Notes to Callers

Compiler overload resolution may account for an apparent difference in the behavior of the two Equals(UInt64) method overloads. If an implicit conversion between the `obj` argument and a UInt64 is defined and the argument is not typed as an Object, compilers perform an implicit conversion and call the Equals(UInt64) method. Otherwise, they call the Equals(Object) method, which always returns `false` if its `obj` argument is not a UInt64 value. The following example illustrates the difference in behavior between the two method overloads. In the case of the Byte, UInt16, and UInt32 values, the first comparison returns `true` because the compiler automatically performs a widening conversion and calls the Equals(UInt64) method, whereas the second comparison returns `false` because the compiler calls the Equals(Object) method.

C#

```
using System;

public class Example
{
   static ulong value = 112;

   public static void Main()
   {
      byte byte1= 112;
      Console.WriteLine("value = byte1: {0,16}", value.Equals(byte1));
```

```
        TestObjectForEquality(byte1);

        short short1 = 112;
        Console.WriteLine("value = short1: {0,17}", value.Equals(short1));
        TestObjectForEquality(short1);

        int int1 = 112;
        Console.WriteLine("value = int1: {0,19}", value.Equals(int1));
        TestObjectForEquality(int1);

        sbyte sbyte1 = 112;
        Console.WriteLine("value = sbyte1: {0,17}", value.Equals(sbyte1));
        TestObjectForEquality(sbyte1);

        ushort ushort1 = 112;
        Console.WriteLine("value = ushort1: {0,16}",
value.Equals(ushort1));
        TestObjectForEquality(ushort1);

        uint uint1 = 112;
        Console.WriteLine("value = uint1: {0,18}", value.Equals(uint1));
        TestObjectForEquality(uint1);

        decimal dec1 = 112m;
        Console.WriteLine("value = dec1: {0,21}", value.Equals(dec1));
        TestObjectForEquality(dec1);

        double dbl1 = 112;
        Console.WriteLine("value = dbl1: {0,20}", value.Equals(dbl1));
        TestObjectForEquality(dbl1);
    }

    private static void TestObjectForEquality(Object obj)
    {
        Console.WriteLine("{0} ({1}) = {2} ({3}): {4}\n",
                          value, value.GetType().Name,
                          obj, obj.GetType().Name,
                          value.Equals(obj));
    }
}
// The example displays the following output:
//       value = byte1:             True
//       112 (UInt64) = 112 (Byte): False
//
//       value = short1:             False
//       112 (UInt64) = 112 (Int16): False
//
//       value = int1:               False
//       112 (UInt64) = 112 (Int32): False
//
//       value = sbyte1:             False
//       112 (UInt64) = 112 (SByte): False
//
//       value = ushort1:             True
//       112 (UInt64) = 112 (UInt16): False
```

```
//
//       value = uint1:                True
//       112 (UInt64) = 112 (UInt32): False
//
//       value = dec1:                  False
//       112 (UInt64) = 112 (Decimal): False
//
//       value = dbl1:                 False
//       112 (UInt64) = 112 (Double): False
```

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt64.GetTypeCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the TypeCode for value type UInt64.

C#

```
public TypeCode GetTypeCode ();
```

Returns

TypeCode

The enumerated constant, UInt64.

Implements

GetTypeCode()

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- TypeCode

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IsEvenInteger(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IsOddInteger(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IsPow2(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.LeadingZeroCount(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

C#

```
public static ulong LeadingZeroCount (ulong value);
```

Parameters

`value` UInt64

The value whose leading zeroes are to be counted.

Returns

UInt64

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Log2(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

C#

```
public static ulong Log2 (ulong value);
```

Parameters

`value` UInt64

The value whose log2 is to be computed.

Returns

UInt64

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Max(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static ulong Max (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Min(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static ulong Min (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the string representation of a number to its 64-bit unsigned integer equivalent.

Overloads

Expand table

Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its 64-bit unsigned integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its 64-bit unsigned integer equivalent.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(String)	Converts the string representation of a number to its 64-bit unsigned integer equivalent.

Parse(String, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Converts the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.

C#

```
public static ulong Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt64

A 64-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

The `s` parameter is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

The `s` parameter is not in a format compliant with `style`.

OverflowException

The `s` parameter represents a number less than UInt64.MinValue or greater than UInt64.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example uses the Parse(String, NumberStyles, IFormatProvider) method to convert various string representations of numbers to 64-bit unsigned integer values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] cultureNames= { "en-US", "fr-FR" };
      NumberStyles[] styles= { NumberStyles.Integer,
                               NumberStyles.Integer |
NumberStyles.AllowDecimalPoint };
      string[] values = { "170209", "+170209.0", "+170209,0",
"-103214.00",
                                  "-103214,00", "104561.1", "104561,1" };

      // Parse strings using each culture
      foreach (string cultureName in cultureNames)
      {
         CultureInfo ci = new CultureInfo(cultureName);
         Console.WriteLine("Parsing strings using the {0} culture",
                           ci.DisplayName);
         // Use each style.
```

```
         foreach (NumberStyles style in styles)
         {
            Console.WriteLine("   Style: {0}", style.ToString());
            // Parse each numeric string.
            foreach (string value in values)
            {
               try {
                  Console.WriteLine("      Converted '{0}' to {1}.",
value,
                                    UInt64.Parse(value, style, ci));
               }
               catch (FormatException) {
                  Console.WriteLine("      Unable to parse '{0}'.",
value);
               }
               catch (OverflowException) {
                  Console.WriteLine("      '{0}' is out of range of the
UInt64 type.",
                                    value);
               }
            }
         }
      }
   }
}
// The example displays the following output:
//       Style: Integer
//          Converted '170209' to 170209.
//          Unable to parse '+170209.0'.
//          Unable to parse '+170209,0'.
//          Unable to parse '-103214.00'.
//          Unable to parse '-103214,00'.
//          Unable to parse '104561.1'.
//          Unable to parse '104561,1'.
//       Style: Integer, AllowDecimalPoint
//          Converted '170209' to 170209.
//          Converted '+170209.0' to 170209.
//          Unable to parse '+170209,0'.
//          '-103214.00' is out of range of the UInt64 type.
//          Unable to parse '-103214,00'.
//          '104561.1' is out of range of the UInt64 type.
//          Unable to parse '104561,1'.
//    Parsing strings using the French (France) culture
//       Style: Integer
//          Converted '170209' to 170209.
//          Unable to parse '+170209.0'.
//          Unable to parse '+170209,0'.
//          Unable to parse '-103214.00'.
//          Unable to parse '-103214,00'.
//          Unable to parse '104561.1'.
//          Unable to parse '104561,1'.
//       Style: Integer, AllowDecimalPoint
//          Converted '170209' to 170209.
//          Unable to parse '+170209.0'.
//          Converted '+170209,0' to 170209.
```

```
//          Unable to parse '-103214.00'.
//          '-103214,00' is out of range of the UInt64 type.
//          Unable to parse '104561.1'.
//          '104561,1' is out of range of the UInt64 type.
```

Remarks

The `style` parameter defines the style elements (such as white space or the positive or negative sign symbol) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object that is returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. (The method throws an OverflowException if `s` includes a negative sign and represents a non-zero number.) The sign can appear at the beginning of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can

Element	Description
	be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag.
digits	A sequence of digits from 0 through 9.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0-9 if `style` includes the NumberStyles.AllowExponent flag, or one or more occurrences of the digit 0 if it does not. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None style) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
NumberStyles.None	Decimal digits only.
NumberStyles.AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, if style does not include the

Non-composite `NumberStyles` values	Elements permitted in `s` in addition to digits
	NumberStyles.AllowExponent flag, *fractional_digits* must consist of only one or more 0 digits; otherwise, an OverflowException is thrown.
NumberStyles.AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
NumberStyles.AllowLeadingWhite	The *ws* element at the beginning of `s`.
NumberStyles.AllowTrailingWhite	The *ws* element at the end of `s`.
NumberStyles.AllowLeadingSign	A sign before *digits*.
NumberStyles.AllowTrailingSign	A sign after *digits*.
NumberStyles.AllowParentheses	Parentheses before and after *digits* to indicate a negative value.
NumberStyles.AllowThousands	The group separator (,) element.
NumberStyles.AllowCurrencySymbol	The currency ($) element.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal characters are 0-9, A-F, and a-f. A prefix such as "0x" is not supported and causes the parse operation to fail. The only other flags that can be combined with it are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

> **Note**
>
> If the `s` parameter is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the parse operation to throw an exception.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.
- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- ToString
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

> Important
>
> This API is not CLS-compliant.

Converts the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent.

```C#
public static ulong Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt64

A 64-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static ulong Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt64

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String, IFormatProvider)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Converts the string representation of a number in a specified culture-specific format to its 64-bit unsigned integer equivalent.

```
C#
public static ulong Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string that represents the number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UInt64

A 64-bit unsigned integer equivalent to the number specified in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

The `s` parameter is `null`.

FormatException

The `s` parameter is not in the correct style.

OverflowException

The `s` parameter represents a number less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example is the button click event handler of a Web form. It uses the array returned by the HttpRequest.UserLanguages property to determine the user's locale. It then instantiates a CultureInfo object that corresponds to that locale. The NumberFormatInfo object that belongs to that CultureInfo object is then passed to the Parse(String, IFormatProvider) method to convert the user's input to a UInt64 value.

C#

```
protected void OkToSingle_Click(object sender, EventArgs e)
{
    string locale;
    float number;
    CultureInfo culture;

    // Return if string is empty
    if (String.IsNullOrEmpty(this.inputNumber.Text))
        return;

    // Get locale of web request to determine possible format of number
    if (Request.UserLanguages.Length == 0)
        return;
    locale = Request.UserLanguages[0];
    if (String.IsNullOrEmpty(locale))
        return;

    // Instantiate CultureInfo object for the user's locale
    culture = new CultureInfo(locale);

    // Convert user input from a string to a number
    try
    {
        number = Single.Parse(this.inputNumber.Text,
culture.NumberFormat);
    }
    catch (FormatException)
    {
        return;
    }
    catch (Exception)
    {
        return;
    }
    // Output number to label on web form
```

```
    this.outputNumber.Text = "Number is " + number.ToString();
}
```

Remarks

This overload of the Parse(String, IFormatProvider) method is typically used to convert text that can be formatted in a variety of ways to a UInt64 value. For example, it can be used to convert the text entered by a user into an HTML text box to a numeric value.

The `s` parameter contains a number of the form:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional positive sign, or a negative sign if `s` represents the value zero.
digits	A sequence of digits ranging from 0 to 9.

The s parameter is interpreted using the NumberStyles.Integer style. In addition to the unsigned integer value's decimal digits, only leading and trailing spaces along with a leading sign is allowed. (If the negative sign is present, `s` must represent a value of zero, or the method throws an OverflowException.) To explicitly define the style elements together with the culture-specific formatting information that can be present in `s`, use the Parse(String, NumberStyles, IFormatProvider) method.

The `provider` parameter is an IFormatProvider implementation whose GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. There are three ways to use the `provider` parameter to supply custom formatting information to the parse operation:

- You can pass the actual NumberFormatInfo object that provides formatting information. (Its implementation of GetFormat simply returns itself.)
- You can pass a CultureInfo object that specifies the culture whose formatting is to be used. Its NumberFormat property provides formatting information.

- You can pass a custom IFormatProvider implementation. Its GetFormat method must instantiate and return the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo for the current culture is used.

See also

- ToString()
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(String, NumberStyles)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Converts the string representation of a number in a specified style to its 64-bit unsigned integer equivalent.

C#

```
public static ulong Parse (string s, System.Globalization.NumberStyles
```

```
style);
```

Parameters

s String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

style NumberStyles

A bitwise combination of the enumeration values that specifies the permitted format of `s`. A typical value to specify is Integer.

Returns

UInt64

A 64-bit unsigned integer equivalent to the number specified in `s`.

Exceptions

ArgumentNullException

The `s` parameter is `null`.

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

The `s` parameter is not in a format compliant with `style`.

OverflowException

The `s` parameter represents a number less than UInt64.MinValue or greater than UInt64.MaxValue.

-or-

`s` includes non-zero, fractional digits.

Examples

The following example tries to parse each element in a string array by using a number of NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string[] values= { " 214309 ", "1,064,181", "(0)", "10241+", " +
21499 ",
                         " +21499 ", "122153.00", "1e03ff", "91300.0e-2"
};
      NumberStyles whitespace =  NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite;
      NumberStyles[] styles= { NumberStyles.None, whitespace,
                               NumberStyles.AllowLeadingSign |
NumberStyles.AllowTrailingSign | whitespace,
                               NumberStyles.AllowThousands |
NumberStyles.AllowCurrencySymbol,
                               NumberStyles.AllowExponent |
NumberStyles.AllowDecimalPoint };

      // Attempt to convert each number using each style combination.
      foreach (string value in values)
      {
         Console.WriteLine("Attempting to convert '{0}':", value);
         foreach (NumberStyles style in styles)
         {
            try {
               ulong number = UInt64.Parse(value, style);
               Console.WriteLine("   {0}: {1}", style, number);
            }
            catch (FormatException) {
               Console.WriteLine("   {0}: Bad Format", style);
            }
            catch (OverflowException)
            {
               Console.WriteLine("   {0}: Overflow", value);
            }
         }
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//    Attempting to convert ' 214309 ':
//       None: Bad Format
```

```
//       AllowLeadingWhite, AllowTrailingWhite: 214309
//       Integer, AllowTrailingSign: 214309
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '1,064,181':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: 1064181
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '(0)':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '10241+':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 10241
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' + 21499 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert ' +21499 ':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: 21499
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '122153.00':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 122153
//
//    Attempting to convert '1e03ff':
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: Bad Format
//
//    Attempting to convert '91300.0e-2':
```

```
//       None: Bad Format
//       AllowLeadingWhite, AllowTrailingWhite: Bad Format
//       Integer, AllowTrailingSign: Bad Format
//       AllowThousands, AllowCurrencySymbol: Bad Format
//       AllowDecimalPoint, AllowExponent: 913
```

Remarks

The `style` parameter defines the style elements (such as white space, the positive or negative sign symbol, the group separator symbol, or the decimal point symbol) that are allowed in the `s` parameter for the parse operation to succeed. `style` must be a combination of bit flags from the NumberStyles enumeration. The `style` parameter makes this method overload useful when `s` contains the string representation of a hexadecimal value, when the number system (decimal or hexadecimal) represented by `s` is known only at run time, or when you want to disallow white space or a sign symbol in `s`.

Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Elements in square brackets ([and]) are optional. If `style` includes NumberStyles.AllowHexSpecifier, the `s` parameter may contain the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the NumberFormatInfo.CurrencyNegativePattern and NumberFormatInfo.CurrencyPositivePattern properties of the current culture. The current culture's currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.

Element	Description
sign	An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits *fractional_digits* *exponential_digits*	A sequence of digits from 0 through 9. For *fractional_digits*, only the digit 0 is valid.
,	A culture-specific group separator symbol. The current culture's group separator can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The current culture's decimal point symbol can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag. Only the digit 0 can appear as a fractional digit for the parse operation to succeed; if *fractional_digits* includes any other digit, a FormatException is thrown.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with digits only (which corresponds to the NumberStyles.None style) always parses successfully if it is in the range of the UInt64 type. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in the input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

`NumberStyles` value	Elements permitted in `s` in addition to digits
None	The *digits* element only.
AllowDecimalPoint	The decimal point (.) and *fractional-digits* elements.
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*.
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element at the start of `s`.
AllowTrailingSign	The *sign* element at the end of `s`.
AllowParentheses	The *sign* element in the form of parentheses enclosing the numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The `ws`, `sign`, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

Unlike the other NumberStyles values, which allow for, but do not require, the presence of particular style elements in `s`, the NumberStyles.AllowHexSpecifier style value means that the individual numeric characters in `s` are always interpreted as hexadecimal characters. Valid hexadecimal characters are 0-9, A-F, and a-f. A prefix such as "0x" is not supported and causes the parse operation to fail. The only other flags that can be combined with the `style` parameter are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration includes a composite number style, NumberStyles.HexNumber, that includes both white-space flags.)

Note

If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `s` parameter is parsed by using the formatting information in a NumberFormatInfo object that is initialized for the current system culture. To specify the culture whose formatting information is used for the parse operation, call the Parse(String, NumberStyles, IFormatProvider) overload.

See also

- ToString
- TryParse
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

C#

```
public static ulong Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UInt64

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static ulong Parse (ReadOnlySpan<byte> utf8Text, IFormatProvider?
provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UInt64

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(String)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Converts the string representation of a number to its 64-bit unsigned integer equivalent.

C#

```
public static ulong Parse (string s);
```

Parameters

s String

A string that represents the number to convert.

Returns

UInt64

A 64-bit unsigned integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

The `s` parameter is `null`.

FormatException

The `s` parameter is not in the correct format.

OverflowException

The `s` parameter represents a number less than UInt64.MinValue or greater than UInt64.MaxValue.

Examples

The following example uses the Parse method to parse an array of string values.

C#

```
string[] values = { "+13230", "-0", "1,390,146", "$190,235,421,127",
                    "0xFA1B", "163042", "-10", "14065839182",
                    "16e07", "134985.0", "-12034" };
foreach (string value in values)
{
   try {
      ulong number = UInt64.Parse(value);
      Console.WriteLine("{0} --> {1}", value, number);
   }
   catch (FormatException) {
      Console.WriteLine("{0}: Bad Format", value);
   }
   catch (OverflowException) {
      Console.WriteLine("{0}: Overflow", value);
   }
}
// The example displays the following output:
//       +13230 --> 13230
//       -0 --> 0
//       1,390,146: Bad Format
//       $190,235,421,127: Bad Format
//       0xFA1B: Bad Format
//       163042 --> 163042
//       -10: Overflow
//       14065839182 --> 14065839182
//       16e07: Bad Format
```

```
//        134985.0: Bad Format
//        -12034: Overflow
```

Remarks

The `s` parameter should be the string representation of a number in the following form.

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture. However, the negative sign symbol can be used only with zero; otherwise, the method throws an OverflowException.
digits	A sequence of digits ranging from 0 to 9. Any leading zeros are ignored.

> **Note**
>
> The string specified by the `s` parameter is interpreted by using the **NumberStyles.Integer** style. It cannot contain any group separators or decimal separator, and it cannot have a decimal portion.

The `s` parameter is parsed by using the formatting information in a System.Globalization.NumberFormatInfo object that is initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo. To parse a string by using the formatting information of a specific culture, use the Parse(String, IFormatProvider) method.

See also

- ToString
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.PopCount(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static ulong PopCount (ulong value);
```

Parameters

`value` UInt64

The value whose set bits are to be counted.

Returns

UInt64

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.RotateLeft(UInt64, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static ulong RotateLeft (ulong value, int rotateAmount);
```

Parameters

`value` UInt64

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

UInt64

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.RotateRight(UInt64, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
public static ulong RotateRight (ulong value, int rotateAmount);
```

Parameters

`value` UInt64

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

UInt64

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.Sign(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (ulong value);
```

Parameters

`value` UInt64

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the numeric value of this instance to its equivalent string representation.

Overloads

Expand table

ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation using the specified format.
ToString(String, IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString()	Converts the numeric value of this instance to its equivalent string representation.

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

provider IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance, consisting of a sequence of digits ranging from 0 to 9, without a sign or leading zeros.

Implements

ToString(IFormatProvider)

Examples

The following example formats a 64-bit signed integer value by using several format providers, including one for the invariant culture. The output from the example illustrates that the formatted string returned by the ToString(IFormatProvider) method is the same regardless of the format provider.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define an array of CultureInfo objects.
      CultureInfo[] ci = { new CultureInfo("en-US"),
                           new CultureInfo("fr-FR"),
                           CultureInfo.InvariantCulture };
      ulong value = 18709243;
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        GetName(ci[0]), GetName(ci[1]), GetName(ci[2]));
      Console.WriteLine("  {0,12}   {1,12}   {2,12}",
                        value.ToString(ci[0]), value.ToString(ci[1]),
value.ToString(ci[2]));
   }

   private static string GetName(CultureInfo ci)
   {
      if (ci.Equals(CultureInfo.InvariantCulture))
         return "Invariant";
      else
         return ci.Name;
   }
}
```

```
// The example displays the following output:
//              en-US          fr-FR      Invariant
//           18709243       18709243       18709243
```

Remarks

The ToString(IFormatProvider) method formats a UInt64 value in the default ("G", or general) format by using the NumberFormatInfo object of a specified culture. If you want to specify a different format or the current culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific formatting information. However, none of the properties of the NumberFormatInfo are used when formatting with the general numeric format specifier ("G").

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

ToString(String)

Converts the numeric value of this instance to its equivalent string representation using the specified format.

```
C#

public string ToString (string? format);
```

Parameters

`format` String

A numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

The `format` parameter is invalid.

Examples

The following example displays a 64-bit unsigned integer value by using each standard format string and some custom format strings.

```
C#

using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
```

```
        ulong value = 217960834;
        string[] specifiers = { "G", "C", "D3", "E2", "e3", "F",
                                "N", "P", "X", "000000.0", "#.0",
                                "00000000;(0);**Zero**" };

        foreach (string specifier in specifiers)
           Console.WriteLine("{0}: {1}", specifier,
value.ToString(specifier));
    }
}
// The example displays the following output:
//       G: 217960834
//       C: $217,960,834.00
//       D3: 217960834
//       E2: 2.18E+008
//       e3: 2.180e+008
//       F: 217960834.00
//       N: 217,960,834.00
//       P: 21,796,083,400.00 %
//       X: CFDD182
//       000000.0: 217960834.0
//       #.0: 217960834.0
//       00000000;(0);**Zero**: 217960834
```

Remarks

The ToString(String) method formats a UInt64 value in a specified format by using a NumberFormatInfo object that represents the conventions of the current culture. If you want to use the default ("G", or general) format or specify a different culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	A specific culture	ToString(String, IFormatProvider)

The `format` parameter can be any valid Standard Numeric Format Strings, or any combination of Custom Numeric Format Strings. If `format` is equal to String.Empty or is `null`, the return value of the current UInt64 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about support for formatting in .NET, see Formatting Types.

The format of the returned string is determined by the NumberFormatInfo object for the current culture. Depending on the `format` parameter, this object controls symbols such as the group separator and the decimal point symbol in the output string. To provide formatting information for cultures other than the current culture, call the ToString(String, IFormatProvider) overload.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(String, IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

```
C#
```

```
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

A numeric format string.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about this instance.

Returns

String

The string representation of the value of this instance as specified by `format` and `provider`.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

The `format` parameter is invalid.

Examples

The following example displays a 32-bit unsigned integer value by using the standard numeric format specifiers and a number of specific CultureInfo objects.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      // Define cultures whose formatting conventions are to be used.
      CultureInfo[] cultures = { CultureInfo.CreateSpecificCulture("en-
US"),
```

```
                                    CultureInfo.CreateSpecificCulture("fr-
FR"),
                                    CultureInfo.CreateSpecificCulture("es-
ES") };
      string[] specifiers = {"G", "C", "D4", "E2", "F", "N", "P", "X2"};
      ulong value = 22224021;

      foreach (string specifier in specifiers)
      {
         foreach (CultureInfo culture in cultures)
            Console.WriteLine("{0,2} format using {1} culture: {2, 18}",
                              specifier, culture.Name,
                              value.ToString(specifier, culture));
         Console.WriteLine();
      }
   }
}
// The example displays the following output:
//        G format using en-US culture:           22224021
//        G format using fr-FR culture:           22224021
//        G format using es-ES culture:           22224021
//
//        C format using en-US culture:     $22,224,021.00
//        C format using fr-FR culture:    22 224 021,00 €
//        C format using es-ES culture:    22.224.021,00 €
//
//       D4 format using en-US culture:           22224021
//       D4 format using fr-FR culture:           22224021
//       D4 format using es-ES culture:           22224021
//
//       E2 format using en-US culture:          2.22E+007
//       E2 format using fr-FR culture:          2,22E+007
//       E2 format using es-ES culture:          2,22E+007
//
//        F format using en-US culture:        22224021.00
//        F format using fr-FR culture:        22224021,00
//        F format using es-ES culture:        22224021,00
//
//        N format using en-US culture:      22,224,021.00
//        N format using fr-FR culture:      22 224 021,00
//        N format using es-ES culture:      22.224.021,00
//
//        P format using en-US culture: 2,222,402,100.00 %
//        P format using fr-FR culture: 2 222 402 100,00 %
//        P format using es-ES culture: 2.222.402.100,00 %
//
//       X2 format using en-US culture:            1531C95
//       X2 format using fr-FR culture:            1531C95
//       X2 format using es-ES culture:            1531C95
```

Remarks

The ToString(String, IFormatProvider) method formats a UInt64 value in a specified format by using the NumberFormatInfo object of a specified culture. If you want to use default format or culture settings, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	Default (current) culture	ToString()
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)

The `format` parameter can be any valid Standard Numeric Format Strings, or any combination of Custom Numeric Format Strings. If `format` is equal to String.Empty or is `null`, the return value of the current UInt64 object is formatted with the general format specifier ("G"). If `format` is any other value, the method throws a FormatException.

.NET provides extensive formatting support, which is described in greater detail in the following formatting topics:

- For more information about numeric format specifiers, see Standard Numeric Format Strings and Custom Numeric Format Strings.
- For more information about formatting, see Formatting Types.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of the string returned by this method. When the ToString(String, IFormatProvider) method is invoked, it calls the `provider` parameter's IFormatProvider.GetFormat method and passes it a Type object that represents the NumberFormatInfo type. The GetFormat method then returns the NumberFormatInfo object that provides information for formatting the current UInt64 value, such as the group separator symbol or the decimal point symbol. There are three ways to use the `provider` parameter to supply formatting information to the ToString(String, IFormatProvider) method:

- You can pass a CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.

- You can pass the actual NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- You can pass a custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the formatting of the returned string is based on the NumberFormatInfo object of the current culture.

See also

- Parse(String)
- Formatting Types in .NET
- How to: Pad a Number with Leading Zeros
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance, consisting of a sequence of digits ranging from 0 to 9, without a sign or leading zeroes.

Examples

The following example displays a UInt64 value by using the default ToString() method. It also displays the string representations of the UInt64 value that results from using some standard format specifiers. The examples are displayed using the formatting conventions of the en-US culture.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      ulong value = 163249057;
      // Display value using default ToString method.
      Console.WriteLine(value.ToString());
      Console.WriteLine();

      // Define an array of format specifiers.
      string[] formats = { "G", "C", "D", "F", "N", "X" };
      // Display value using the standard format specifiers.
      foreach (string format in formats)
         Console.WriteLine("{0} format specifier: {1,16}",
                           format, value.ToString(format));
   }
}
// The example displays the following output:
//       163249057
//
//       G format specifier:        163249057
//       C format specifier:  $163,249,057.00
//       D format specifier:        163249057
//       F format specifier:     163249057.00
//       N format specifier:   163,249,057.00
//       X format specifier:          9BAFBA1
```

Remarks

The ToString() method formats a UInt64 value in the default ("G", or general) format by using the NumberFormatInfo object of the current culture. If you want to specify a different format or culture, use the other overloads of the ToString method, as follows:

Expand table

To use format	For culture	Use the overload
Default ("G") format	A specific culture	ToString(IFormatProvider)
A specific format	Default (current) culture	ToString(String)
A specific format	A specific culture	ToString(String, IFormatProvider)

See also

- Parse(String)
- Formatting Types in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UInt64.TrailingZeroCount(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

C#

```
public static ulong TrailingZeroCount (ulong value);
```

Parameters

`value` UInt64

The value whose trailing zeroes are to be counted.

Returns

UInt64

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current unsigned long number instance into the provided span of characters.

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

utf8Destination Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

bytesWritten Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current unsigned long number instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format of `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

See also

- Standard numeric format strings
- Custom numeric format strings

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert the string representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

Overloads

Expand table

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, UInt64)	Tries to convert the span representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(String, UInt64)	Tries to convert the string representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt64)	Tries to parse a span of characters into a value.
TryParse(String, IFormatProvider, UInt64)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt64)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UInt64)	Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt64)	Tries to convert the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(String, NumberStyles, IFormatProvider, UInt64)	Tries to convert the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UInt64)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out ulong result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt64

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, UInt64)

> Important
>
> This API is not CLS-compliant.

Tries to convert the span representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (ReadOnlySpan<char> s, out ulong result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert.

`result` UInt64

When this method returns, contains the 64-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than UInt64.MinValue or greater than UInt64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, UInt64)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Tries to convert the string representation of a number to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

```
C#
public static bool TryParse (string? s, out ulong result);
```

Parameters

`s` String

A string that represents the number to convert.

`result` UInt64

When this method returns, contains the 64-bit unsigned integer value that is equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not of the correct format, or represents a number less than UInt64.MinValue or greater than UInt64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Examples

The following example calls the TryParse(String, UInt64) method once for each element in a string array.

```
C#
```

```
string[] numericStrings = { "1293.8", "+1671.7", "28347.",
                            "   33113684  ", "(0)", "-0", "+1293617",
                            "18-", "119870", "31,024", "  3127094 ",
                            "00700000" };
ulong number;
foreach (string numericString in numericStrings)
{
   if (UInt64.TryParse(numericString, out number))
      Console.WriteLine("Converted '{0}' to {1}.", numericString,
number);
   else
      Console.WriteLine("Cannot convert '{0}' to a UInt64.",
numericString);
}
// The example displays the following output:
//       Cannot convert '1293.8' to a UInt64.
//       Cannot convert '+1671.7' to a UInt64.
//       Cannot convert '28347.' to a UInt64.
//       Converted '   33113684  ' to 33113684.
//       Cannot convert '(0)' to a UInt64.
//       Converted '-0' to 0.
//       Converted '+1293617' to 1293617.
//       Cannot convert '18-' to a UInt64.
//       Converted '119870' to 119870.
//       Cannot convert '31,024' to a UInt64.
//       Converted '  3127094 ' to 3127094.
//       Converted '0070000' to 70000.
```

Remarks

The TryParse(String, UInt64) method is like the Parse(String) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be successfully parsed.

The `s` parameter should be the string representation of a decimal number in the following form:

[*ws*][*sign*]*digits*[*ws*]

Elements in square brackets ([and]) are optional. The following table describes each element.

Expand table

Element	Description
ws	Optional white space.

Element	Description
sign	An optional sign. Valid sign characters are determined by the NumberFormatInfo.NegativeSign and NumberFormatInfo.PositiveSign properties of the current culture.
digits	A sequence of decimal digits ranging from 0 to 9.

The `s` parameter is interpreted by using the NumberStyles.Integer style. In addition to the decimal digits, only leading and trailing spaces with a leading sign are allowed. To explicitly define the style elements with the culture-specific formatting information that can be present in `s`, call the TryParse(String, NumberStyles, IFormatProvider, UInt64) method.

> **Note**
>
> The string specified by the `s` parameter cannot contain any group separators or decimal separator, and it cannot have a fractional portion.

The `s` parameter is parsed using the formatting information in a NumberFormatInfo object initialized for the current system culture. For more information, see NumberFormatInfo.CurrentInfo.

This overload interprets all digits in the `s` parameter as decimal digits. To parse the string representation of a hexadecimal number, call the TryParse(String, NumberStyles, IFormatProvider, UInt64) overload instead.

See also

- Parse
- ToString
- Parsing Numeric Strings in .NET
- Sample: .NET Core WinForms Formatting Utility (C#)
- Sample: .NET Core WinForms Formatting Utility (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, IFormatProvider, UInt64)

Tries to parse a span of characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out ulong result);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt64

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(String, IFormatProvider, UInt64)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
ulong result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UInt64

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UInt64)

Tries to parse a span of UTF-8 characters into a value.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
ulong result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UInt64

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Byte>, UInt64)

Tries to convert a UTF-8 character span containing the string representation of a number to its 64-bit unsigned integer equivalent.

```
C#

public static bool TryParse (ReadOnlySpan<byte> utf8Text, out ulong
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` UInt64

When this method returns, contains the 64-bit unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UInt64)

Important

This API is not CLS-compliant.

Tries to convert the span representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
ulong result);
```

Parameters

`s` ReadOnlySpan<Char>

A span containing the characters that represent the number to convert. The span is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt64

When this method returns, contains the 64-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than UInt64.MinValue or greater than UInt64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryParse(String, NumberStyles, IFormatProvider, UInt64)

> Important
>
> This API is not CLS-compliant.
>
> CLS-compliant alternative **System.Decimal.Parse(String)**

Tries to convert the string representation of a number in a specified style and culture-specific format to its 64-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out ulong result);
```

Parameters

`s` String

A string that represents the number to convert. The string is interpreted by using the style specified by the `style` parameter.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the permitted format of `s`. A typical value to specify is Integer.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UInt64

When this method returns, contains the 64-bit unsigned integer value equivalent to the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or Empty, is not in a format compliant with `style`, or represents a number less than UInt64.MinValue or greater than UInt64.MaxValue. This parameter is passed uninitialized; any value originally supplied in `result` will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Exceptions

ArgumentException

`style` is not a NumberStyles value.

-or-

`style` is not a combination of AllowHexSpecifier and HexNumber values.

Examples

The following example calls the TryParse(String, NumberStyles, IFormatProvider, UInt64) method with a number of different strings and NumberStyles values.

C#

```
using System;
using System.Globalization;

public class Example
{
   public static void Main()
   {
      string numericString;
      NumberStyles styles;

      numericString = "2106034";
      styles = NumberStyles.Integer;
      CallTryParse(numericString, styles);

      numericString = "-10603";
      styles = NumberStyles.None;
      CallTryParse(numericString, styles);
```

```
      numericString = "29103674.00";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "10345.72";
      styles = NumberStyles.Integer | NumberStyles.AllowDecimalPoint;
      CallTryParse(numericString, styles);

      numericString = "41792210E-01";
      styles = NumberStyles.Integer | NumberStyles.AllowExponent;
      CallTryParse(numericString, styles);

      numericString = "9112E-01";
      CallTryParse(numericString, styles);

      numericString = "312E01";
      CallTryParse(numericString, styles);

      numericString = "FFC86DA1";
      CallTryParse(numericString, NumberStyles.HexNumber);

      numericString = "0x8F8C";
      CallTryParse(numericString, NumberStyles.HexNumber);
   }

   private static void CallTryParse(string stringToConvert, NumberStyles
styles)
   {
      ulong number;
      bool result = UInt64.TryParse(stringToConvert, styles,
                                    CultureInfo.InvariantCulture, out
number);
      if (result)
         Console.WriteLine($"Converted '{stringToConvert}' to
{number}.");
      else
         Console.WriteLine($"Attempted conversion of '{stringToConvert}'
failed.");
   }
}
// The example displays the following output:
//       Converted '2106034' to 2106034.
//       Attempted conversion of '-10603' failed.
//       Converted '29103674.00' to 29103674.
//       Attempted conversion of '10345.72' failed.
//       Converted '41792210E-01' to 4179221.
//       Attempted conversion of '9112E-01' failed.
//       Converted '312E01' to 3120.
//       Converted 'FFC86DA1' to 4291325345.
//       Attempted conversion of '0x8F8C' failed.
```

Remarks

The TryParse(String, NumberStyles, IFormatProvider, UInt64) method is like the Parse(String, NumberStyles, IFormatProvider) method, except that it does not throw an exception if the conversion fails. This method eliminates the need to use exception handling to test for a FormatException if `s` is invalid and cannot be parsed successfully.

The `style` parameter defines the style elements (such as white space or a positive or negative sign) that are allowed in the `s` parameter for the parse operation to succeed. It must be a combination of bit flags from the NumberStyles enumeration. Depending on the value of `style`, the `s` parameter may include the following elements:

[*ws*][*$*][*sign*][*digits*,]*digits*[.*fractional_digits*][E[*sign*]*exponential_digits*][*ws*]

Items in square brackets ([and]) are optional. Or, if the `style` parameter includes NumberStyles.AllowHexSpecifier, the `s` parameter may include the following elements:

[*ws*]*hexdigits*[*ws*]

The following table describes each element.

Expand table

Element	Description
ws	Optional white space. White space can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingWhite flag, or at the end of `s` if `style` includes the NumberStyles.AllowTrailingWhite flag.
$	A culture-specific currency symbol. Its position in the string is defined by the CurrencyPositivePattern property of the NumberFormatInfo object returned by the GetFormat method of the `provider` parameter. The currency symbol can appear in `s` if `style` includes the NumberStyles.AllowCurrencySymbol flag.
sign	An optional sign. The sign can appear at the start of `s` if `style` includes the NumberStyles.AllowLeadingSign flag, and it can appear at the end of `s` if `style` includes the NumberStyles.AllowTrailingSign flag. Parentheses can be used in `s` to indicate a negative value if `style` includes the NumberStyles.AllowParentheses flag. However, if the negative sign is present, `s` can only represent the value zero for the parse operation to succeed.
digits	A sequence of digits from 0 through 9.

Element	Description
,	A culture-specific group separator. The group separator of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowThousands flag.
.	A culture-specific decimal point symbol. The decimal point symbol of the culture specified by `provider` can appear in `s` if `style` includes the NumberStyles.AllowDecimalPoint flag.
fractional_digits	One or more occurrences of the digit 0. Fractional digits can appear in `s` only if `style` includes the NumberStyles.AllowDecimalPoint flag.
E	The "e" or "E" character, which indicates that the value is represented in exponential (scientific) notation. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
exponential_digits	A sequence of digits from 0 through 9. The `s` parameter can represent a number in exponential notation if `style` includes the NumberStyles.AllowExponent flag.
hexdigits	A sequence of hexadecimal digits from 0 through f, or 0 through F.

> **Note**
>
> Any terminating NUL (U+0000) characters in `s` are ignored by the parsing operation, regardless of the value of the `style` argument.

A string with decimal digits only (which corresponds to the NumberStyles.None flag) always parses successfully. Most of the remaining NumberStyles members control elements that may be present, but are not required to be present, in this input string. The following table indicates how individual NumberStyles members affect the elements that may be present in `s`.

Expand table

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
None	Decimal digits only.
AllowDecimalPoint	The decimal point (.) and *fractional_digits* elements. However, *fractional_digits* must consist of only one or more 0 digits, or the method returns `false`.

Non-composite `NumberStyles` values	Elements permitted in value in addition to digits
AllowExponent	The "e" or "E" character, which indicates exponential notation, along with *exponential_digits*. If `s` represents a number in exponential notation, it cannot have a non-zero, fractional component.
AllowLeadingWhite	The *ws* element at the start of `s`.
AllowTrailingWhite	The *ws* element at the end of `s`.
AllowLeadingSign	The *sign* element before *digits*.
AllowTrailingSign	The *sign* element after *digits*.
AllowParentheses	The *sign* element in the form of parentheses enclosing a zero numeric value.
AllowThousands	The group separator (,) element.
AllowCurrencySymbol	The currency ($) element.
Currency	All elements. However, `s` cannot represent a hexadecimal number or a number in exponential notation.
Float	The *ws* element at the start or end of `s`, *sign* at the start of `s`, and the decimal point (.) symbol. The `s` parameter can also use exponential notation.
Number	The *ws*, *sign*, group separator (,), and decimal point (.) elements.
Any	All elements. However, `s` cannot represent a hexadecimal number.

If the NumberStyles.AllowHexSpecifier flag is used, `s` must be a hexadecimal value. Valid hexadecimal characters are 0-9, A-F, and a-f. A prefix such as "0x" is not supported and causes the parse operation to fail. The only other flags that can be present in `style` are NumberStyles.AllowLeadingWhite and NumberStyles.AllowTrailingWhite. (The NumberStyles enumeration has a composite style, HexNumber, that includes both white-space flags.)

> **Note**
>
> If `s` is the string representation of a hexadecimal number, it cannot be preceded by any decoration (such as `0x` or `&h`) that differentiates it as a hexadecimal number. This causes the conversion to fail.

The `provider` parameter is an IFormatProvider implementation. Its GetFormat method returns a NumberFormatInfo object that provides culture-specific information about the format of `s`. The `provider` parameter can be any one of the following:

- A CultureInfo object that represents the culture that supplies formatting information. Its GetFormat method returns the NumberFormatInfo object that provides numeric formatting information for that culture.
- A NumberFormatInfo object that provides numeric formatting information. (Its implementation of GetFormat just returns itself.)
- A custom object that implements IFormatProvider. Its GetFormat method instantiates and returns the NumberFormatInfo object that provides formatting information.

If `provider` is `null`, the NumberFormatInfo object for the current culture is used.

See also

- Parse(String)
- Parse
- ToString
- Parsing Numeric Strings in .NET

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IConvertible.ToBoolean(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToBoolean(IFormatProvider).

```
C#
bool IConvertible.ToBoolean (IFormatProvider provider);
```

Parameters

`provider` IFormatProvider

This parameter is ignored.

Returns

Boolean

`true` if the value of the current instance is not zero; otherwise, `false`.

Implements

ToBoolean(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToBoolean(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToByte(IFormatProvider).

```
C#
byte IConvertible.ToByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Byte

The value of the current instance, converted to a Byte.

Implements

ToByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToByte(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToChar(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToChar(IFormatProvider).

C#

```
char IConvertible.ToChar (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Char

The value of the current instance, converted to a Char.

Implements

ToChar(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToChar(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToDateTime(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

This conversion is not supported. Attempting to use this method throws an InvalidCastException.

```
C#

DateTime IConvertible.ToDateTime (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

DateTime

This conversion is not supported. No value is returned.

Implements

ToDateTime(IFormatProvider)

Exceptions

InvalidCastException

In all cases.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToDecimal(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDecimal(IFormatProvider).

C#

```
decimal IConvertible.ToDecimal (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Decimal

The value of the current instance, converted to a Decimal.

Implements

ToDecimal(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDecimal(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToDouble(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToDouble(IFormatProvider).

```
C#
double IConvertible.ToDouble (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Double

The value of the current instance, converted to a Double.

Implements

ToDouble(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToDouble(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt16(IFormatProvider).

C#

```
short IConvertible.ToInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int16

The value of the current instance, converted to an Int16.

Implements

ToInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt16(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt32(IFormatProvider).

```
C#
int IConvertible.ToInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int32

The value of the current instance, converted to an Int32.

Implements

ToInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt32(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToInt64(IFormatProvider).

```
C#
long IConvertible.ToInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Int64

The value of the current instance, converted to an Int64.

Implements

ToInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToInt64(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToSByte(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSByte(IFormatProvider).

```
C#
sbyte IConvertible.ToSByte (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

SByte

The value of the current instance, converted to an SByte.

Implements

ToSByte(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSByte(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToSingle(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToSingle(IFormatProvider).

```
C#
float IConvertible.ToSingle (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

Single

The value of the current instance, converted to a Single.

Implements

ToSingle(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the Convert.ToSingle(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToType(Type, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

For a description of this member, see ToType(Type, IFormatProvider).

C#

```
object IConvertible.ToType (Type type, IFormatProvider provider);
```

Parameters

type Type

The type to which to convert this UInt64 value.

provider IFormatProvider

An IFormatProvider implementation that supplies information about the format of the returned value.

Returns

Object

The value of the current instance, converted to `type`.

Implements

ToType(Type, IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is to call the `static` (`Shared` in Visual Basic) Convert.ChangeType method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.To UInt16(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt16(IFormatProvider).

C#

```
ushort IConvertible.ToUInt16 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt16

The value of the current instance, converted to a UInt16.

Implements

ToUInt16(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt16(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToUInt32(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt32(IFormatProvider).

C#

```
uint IConvertible.ToUInt32 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt32

The value of the current instance, converted to a UInt32.

Implements

ToUInt32(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface. The recommended alternative is

to call the Convert.ToUInt32(UInt64) method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IConvertible.ToUInt64(IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

For a description of this member, see ToUInt64(IFormatProvider).

C#

```
ulong IConvertible.ToUInt64 (IFormatProvider provider);
```

Parameters

provider IFormatProvider

This parameter is ignored.

Returns

UInt64

The value of the current instance, unchanged.

Implements

ToUInt64(IFormatProvider)

Remarks

This member is an explicit interface member implementation. It can be used only when the UInt64 instance is cast to an IConvertible interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.IAdditionOperators<UInt64,UInt64,UInt64>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static ulong IAdditionOperators<ulong,ulong,ulong>.operator + (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to which `right` is added.

`right` UInt64

The value that's added to `left`.

Returns

UInt64

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IAdditionOperators<UInt64,UInt64,UInt64>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static ulong IAdditionOperators<ulong,ulong,ulong>.op_CheckedAddition (ulong
left, ulong right);
```

Parameters

`left` UInt64

The value to which `right` is added.

`right` UInt64

The value that's added to `left`.

Returns

UInt64

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IAdditiveIdentity<UInt64,UInt64>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static ulong
System.Numerics.IAdditiveIdentity<System.UInt64,System.UInt64>.AdditiveIdentity { get; }
```

Property Value

UInt64

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt64.IBinaryInteger<UInt64>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

```
C#

int IBinaryInteger<ulong>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt64.IBinaryInteger<UInt64>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<ulong>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt64.IBinaryInteger<UInt64>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<ulong>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out ulong value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt64.IBinaryInteger<UInt64>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<ulong>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out ulong value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UInt64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UInt64.IBinaryInteger<UInt64>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<ulong>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IBinaryInteger<UInt64>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<ulong>.TryWriteLittleEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IBinaryNumber<UInt64>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static ulong System.Numerics.IBinaryNumber<System.UInt64>.AllBitsSet { get;
}
```

Property Value

UInt64

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt64.IBitwiseOperators<UInt64,UInt64,UInt64>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

```
C#

static ulong IBitwiseOperators<ulong,ulong,ulong>.operator & (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to `and` with `right`.

`right` UInt64

The value to `and` with `left`.

Returns

UInt64

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IBitwiseOperators<UInt64,UInt64,UInt64>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

```
C#

static ulong IBitwiseOperators<ulong,ulong,ulong>.operator | (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to `or` with `right`.

`right` UInt64

The value to `or` with `left`.

Returns

UInt64

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IBitwiseOperators<UInt64,UInt64,UInt64>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static ulong IBitwiseOperators<ulong,ulong,ulong>.operator ^ (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to xor with `right`.

`right` UInt64

The value to xor with `left`.

Returns

UInt64

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IBitwiseOperators<UInt64,UInt64,UInt64>.OnesComplementOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

```
C#
static ulong IBitwiseOperators<ulong,ulong,ulong>.operator ~ (ulong value);
```

Parameters

`value` UInt64

The value for which to compute the ones-complement.

Returns

UInt64

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IComparisonOperators<UInt64,UInt64,Boolean>.GreaterThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<ulong,ulong,bool>.operator > (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IComparisonOperators<UInt64,UInt64,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<ulong,ulong,bool>.operator >= (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IComparisonOperators<UInt64,UInt64,Boolean>.LessThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<ulong,ulong,bool>.operator < (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IComparisonOperators<UInt64,UInt64,Boolean>.LessThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<ulong,ulong,bool>.operator <= (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IDecrementOperators<UInt64>.CheckedDecrement(UInt64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static ulong IDecrementOperators<ulong>.op_CheckedDecrement (ulong value);
```

Parameters

`value` UInt64

The value to decrement.

Returns

UInt64

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IDecrementOperators<UInt64>.Decrement(UInt64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#
static ulong IDecrementOperators<ulong>.operator -- (ulong value);
```

Parameters

`value` UInt64

The value to decrement.

Returns

UInt64

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IDivisionOperators<UInt64,UInt64,UInt64>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static ulong IDivisionOperators<ulong,ulong,ulong>.operator / (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value that `right` divides.

`right` UInt64

The value that divides `left`.

Returns

UInt64

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IEqualityOperators<UInt64,UInt64,Boolean>.Equality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine equality.

```
C#

static bool IEqualityOperators<ulong,ulong,bool>.operator == (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is equal to `right`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IEqualityOperators<UInt64,UInt64,Boolean>.Inequality Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine inequality.

```
C#

static bool IEqualityOperators<ulong,ulong,bool>.operator != (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value to compare with `right`.

`right` UInt64

The value to compare with `left`.

Returns

Boolean

`true` if `left` is not equal to `right`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IIncrementOperators<UInt64>.CheckedIncrement(UInt64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static ulong IIncrementOperators<ulong>.op_CheckedIncrement (ulong value);
```

Parameters

`value` UInt64

The value to increment.

Returns

UInt64

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IIncrementOperators<UInt64>.Increment(UInt64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static ulong IIncrementOperators<ulong>.operator ++ (ulong value);
```

Parameters

`value` UInt64

The value to increment.

Returns

UInt64

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IMinMaxValue<UInt64>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static ulong System.Numerics.IMinMaxValue<System.UInt64>.MaxValue { get; }
```

Property Value

UInt64

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IMinMaxValue<UInt64>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static ulong System.Numerics.IMinMaxValue<System.UInt64>.MinValue { get; }
```

Property Value

UInt64

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IModulusOperators<UInt64,UInt64,UInt64>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static ulong IModulusOperators<ulong,ulong,ulong>.operator % (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value that `right` divides.

`right` UInt64

The value that divides `left`.

Returns

UInt64

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IMultiplicativeIdentity<UInt64,UInt64>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static ulong
System.Numerics.IMultiplicativeIdentity<System.UInt64,System.UInt64>.MultiplicativeIdentity { get; }
```

Property Value

UInt64

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt64.IMultiplyOperators<UInt64,UInt64,UInt64>.CheckedMultiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static ulong IMultiplyOperators<ulong,ulong,ulong>.op_CheckedMultiply (ulong
left, ulong right);
```

Parameters

`left` UInt64

The value that `right` multiplies.

`right` UInt64

The value that multiplies `left`.

Returns

UInt64

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IMultiplyOperators<UInt64,UInt64,UInt64>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static ulong IMultiplyOperators<ulong,ulong,ulong>.operator * (ulong left,
ulong right);
```

Parameters

`left` UInt64

The value that `right` multiplies.

`right` UInt64

The value that multiplies `left`.

Returns

UInt64

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumber<UInt64>.CopySign(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static ulong INumber<ulong>.CopySign (ulong value, ulong sign);
```

Parameters

`value` UInt64

The value whose magnitude is used in the result.

`sign` UInt64

The value whose sign is used in the result.

Returns

UInt64

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumber<UInt64>.MaxNumber(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

C#

```
static ulong INumber<ulong>.MaxNumber (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumber<UInt64>.MinNumber(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static ulong INumber<ulong>.MinNumber (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.Abs(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static ulong INumberBase<ulong>.Abs (ulong value);
```

Parameters

`value` UInt64

The value for which to get its absolute.

Returns

UInt64

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsCanonical(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

```
C#
static bool INumberBase<ulong>.IsCanonical (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsComplexNumber(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

C#

```
static bool INumberBase<ulong>.IsComplexNumber (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsFinite(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<ulong>.IsFinite (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsImaginaryNumber(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<ulong>.IsImaginaryNumber (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsInfinity(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<ulong>.IsInfinity (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsInteger(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<ulong>.IsInteger (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsNaN(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<ulong>.IsNaN (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsNegative(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<ulong>.IsNegative (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsNegativeInfinity(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<ulong>.IsNegativeInfinity (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsNormal(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<ulong>.IsNormal (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsPositive(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

```
C#
static bool INumberBase<ulong>.IsPositive (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsPositiveInfinity(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<ulong>.IsPositiveInfinity (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsRealNumber(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

```
C#
static bool INumberBase<ulong>.IsRealNumber (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsSubnormal(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<ulong>.IsSubnormal (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.IsZero(UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<ulong>.IsZero (ulong value);
```

Parameters

`value` UInt64

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.MaxMagnitude(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static ulong INumberBase<ulong>.MaxMagnitude (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.MaxMagnitudeNumber(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

```
C#

static ulong INumberBase<ulong>.MaxMagnitudeNumber (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.MinMagnitude(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
static ulong INumberBase<ulong>.MinMagnitude (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.MinMagnitudeNumber(UInt64, UInt64) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

```
C#

static ulong INumberBase<ulong>.MinMagnitudeNumber (ulong x, ulong y);
```

Parameters

x UInt64

The value to compare with `y`.

y UInt64

The value to compare with `x`.

Returns

UInt64

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static ulong System.Numerics.INumberBase<System.UInt64>.One { get; }
```

Property Value

UInt64

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt64.INumberBase<UInt64>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.UInt64>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ulong>.TryConvertFromChecked<TOther> (TOther value,
out ulong result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ulong>.TryConvertFromSaturating<TOther> (TOther
value, out ulong result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<ulong>.TryConvertFromTruncating<TOther> (TOther
value, out ulong result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UInt64

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<ulong>.TryConvertToChecked<TOther> (ulong value, out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<ulong>.TryConvertToSaturating<TOther> (ulong value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

C#

```
static bool INumberBase<ulong>.TryConvertToTruncating<TOther> (ulong value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UInt64

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.INumberBase<UInt64>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static ulong System.Numerics.INumberBase<System.UInt64>.Zero { get; }
```

Property Value

UInt64

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UInt64.IShift Operators<UInt64,Int32,UInt64>.Left Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

```
C#

static ulong IShiftOperators<ulong,int,ulong>.operator << (ulong value, int
shiftAmount);
```

Parameters

`value` UInt64

The value that is shifted left by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted left.

Returns

UInt64

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IShift Operators<UInt64,Int32,UInt64>.Right Shift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static ulong IShiftOperators<ulong,int,ulong>.operator >> (ulong value, int
shiftAmount);
```

Parameters

`value` UInt64

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt64

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IShift Operators<UInt64,Int32,UInt64>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static ulong IShiftOperators<ulong,int,ulong>.op_UnsignedRightShift (ulong
value, int shiftAmount);
```

Parameters

`value` UInt64

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UInt64

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.ISubtraction Operators<UInt64,UInt64,UInt64>.CheckedSubtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static ulong ISubtractionOperators<ulong,ulong,ulong>.op_CheckedSubtraction
(ulong left, ulong right);
```

Parameters

`left` UInt64

The value from which `right` is subtracted.

`right` UInt64

The value that's subtracted from `left`.

Returns

UInt64

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.ISubtractionOperators<UInt64,UInt64,UInt64>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static ulong ISubtractionOperators<ulong,ulong,ulong>.operator - (ulong
left, ulong right);
```

Parameters

`left` UInt64

The value from which `right` is subtracted.

`right` UInt64

The value that's subtracted from `left`.

Returns

UInt64

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IUnaryNegationOperators<UInt64,UInt64>.CheckedUnaryNegation Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static ulong IUnaryNegationOperators<ulong,ulong>.op_CheckedUnaryNegation
(ulong value);
```

Parameters

`value` UInt64

The value for which to compute the unary negation.

Returns

UInt64

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IUnaryNegationOperators<UInt64,UInt64>.UnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#
static ulong IUnaryNegationOperators<ulong,ulong>.operator - (ulong value);
```

Parameters

`value` UInt64

The value for which to compute the unary negation.

Returns

UInt64

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UInt64.IUnaryPlusOperators<UInt64,UInt64>.UnaryPlus(UInt64) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#
static ulong IUnaryPlusOperators<ulong,ulong>.operator + (ulong value);
```

Parameters

`value` UInt64

The value for which to compute the unary plus.

Returns

UInt64

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents an unsigned integer where the bit-width is the same as a pointer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UIntPtr : IComparable<UIntPtr>, IEquatable<UIntPtr>,
IParsable<UIntPtr>, ISpanParsable<UIntPtr>, IUtf8SpanParsable<UIntPtr>,
System.Numerics.IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IAdditiveIdentity<UIntPtr,UIntPtr>,
System.Numerics.IBinaryInteger<UIntPtr>,
System.Numerics.IBinaryNumber<UIntPtr>,
System.Numerics.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IComparisonOperators<UIntPtr,UIntPtr,bool>,
System.Numerics.IDecrementOperators<UIntPtr>,
System.Numerics.IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IEqualityOperators<UIntPtr,UIntPtr,bool>,
System.Numerics.IIncrementOperators<UIntPtr>,
System.Numerics.IMinMaxValue<UIntPtr>,
System.Numerics.IModulusOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IMultiplicativeIdentity<UIntPtr,UIntPtr>,
System.Numerics.IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.INumber<UIntPtr>, System.Numerics.INumberBase<UIntPtr>,
System.Numerics.IShiftOperators<UIntPtr,int,UIntPtr>,
System.Numerics.ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IUnaryNegationOperators<UIntPtr,UIntPtr>,
System.Numerics.IUnaryPlusOperators<UIntPtr,UIntPtr>,
System.Numerics.IUnsignedNumber<UIntPtr>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → UIntPtr

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UIntPtr> , IEquatable<UIntPtr> , IFormattable , ISerializable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<UIntPtr> , IParsable<TSelf> , ISpanParsable<UIntPtr> , ISpanParsable<TSelf> , IAdditionOperators<UIntPtr,UIntPtr,UIntPtr> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UIntPtr,UIntPtr> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UIntPtr> , IBinaryNumber<UIntPtr> , IBinaryNumber<TSelf> , IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UIntPtr,UIntPtr,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UIntPtr> , IDecrementOperators<TSelf> , IDivisionOperators<UIntPtr,UIntPtr,UIntPtr> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UIntPtr,UIntPtr,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UIntPtr> , IIncrementOperators<TSelf> , IMinMaxValue<UIntPtr> , IModulusOperators<UIntPtr,UIntPtr,UIntPtr> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UIntPtr,UIntPtr> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UIntPtr> , INumber<TSelf> , INumberBase<UIntPtr> , INumberBase<TSelf> , IShiftOperators<UIntPtr,Int32,UIntPtr> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UIntPtr,UIntPtr> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<UIntPtr,UIntPtr> , IUnaryPlusOperators<TSelf,TSelf> ,

IUnsignedNumber<UIntPtr> , IUtf8SpanFormattable ,

IUtf8SpanParsable<UIntPtr> , IUtf8SpanParsable<TSelf>

Remarks

The UIntPtr type is designed to be an integer whose size is the same as a pointer. That is, an instance of this type is expected to be 32-bits in a 32-bit process and 64-bits in a 64-bit process.

The UIntPtr type can be used by languages that support pointers, and as a common means of referring to data between languages that do and do not support pointers. UIntPtr objects can also be used to hold handles.

> **Note**
>
> Using **UIntPtr** as a pointer or a handle is error prone and unsafe. It is simply an integer type that can be used as an interchange format for pointers and handles due to being the same size. Outside of specific interchange requirements, such as for passing data to a language that doesn't support pointers, a correctly typed pointer should be used to represent pointers and **SafeHandle** should be used to represent handles.

This type implements the ISerializable. In .NET 5 and later versions, this type also implements the IFormattable interfaces. In .NET 7 and later versions, this type also implements the IBinaryInteger<TSelf>, IMinMaxValue<TSelf>, and IUnsignedNumber<TSelf> interfaces.

In C# starting from version 9.0, you can use the built-in `nuint` type to define native-sized integers. This type is represented by the UIntPtr type internally and provides operations and conversions that are appropriate for integer types. For more information, see nint and nuint types.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, `nuint` is an alias for UIntPtr in the same way that `uint` is an alias for UInt32.

Constructors

Expand table

UInt Ptr(UInt32)	Initializes a new instance of the UIntPtr structure using the specified 32-bit unsigned integer.
UInt Ptr(UInt64)	Initializes a new instance of UIntPtr using the specified 64-bit unsigned integer.
UIntPtr(Void*)	Initializes a new instance of UIntPtr using the specified pointer to an unspecified type.

Fields

Expand table

Zero	A read-only field that represents an unsigned integer that has been initialized to zero.

Properties

Expand table

MaxValue	Represents the largest possible value of UIntPtr.
MinValue	Represents the smallest possible value of UIntPtr.
Size	Gets the size of this instance.

Methods

Expand table

Add(UIntPtr, Int32)	Adds an offset to an unsigned integer.
Clamp(UIntPtr, UIntPtr, UIntPtr)	Clamps a value to an inclusive minimum and maximum value.
CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(UIntPtr)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

CreateChecked<TOther>(TOther)	Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.
CreateSaturating<TOther>(TOther)	Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.
CreateTruncating<TOther>(TOther)	Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.
DivRem(UIntPtr, UIntPtr)	Computes the quotient and remainder of two values.
Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UIntPtr)	Indicates whether the current object is equal to another object of the same type.
GetHashCode()	Returns the hash code for this instance.
IsEvenInteger(UIntPtr)	Determines if a value represents an even integral number.
IsOddInteger(UIntPtr)	Determines if a value represents an odd integral number.
IsPow2(UIntPtr)	Determines if a value is a power of two.
LeadingZeroCount(UIntPtr)	Computes the number of leading zeros in a value.
Log2(UIntPtr)	Computes the log2 of a value.
Max(UIntPtr, UIntPtr)	Compares two values to compute which is greater.
Min(UIntPtr, UIntPtr)	Compares two values to compute which is lesser.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the read-only span of characters representation of a number in optionally specified style and optionally specified culture-specific format to its unsigned native integer equivalent.
Parse(String)	Converts the string representation of a number to its unsigned native integer equivalent.

Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its unsigned native integer equivalent.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its unsigned native integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent.
PopCount(UIntPtr)	Computes the number of bits that are set in a value.
RotateLeft(UIntPtr, Int32)	Rotates a value left by a given amount.
RotateRight(UIntPtr, Int32)	Rotates a value right by a given amount.
Sign(UIntPtr)	Computes the sign of a value.
Subtract(UIntPtr, Int32)	Subtracts an offset from an unsigned integer.
ToPointer()	Converts the value of this instance to a pointer to an unspecified type.
ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
ToUInt32()	Converts the value of this instance to a 32-bit unsigned integer.
ToUInt64()	Converts the value of this instance to a 64-bit unsigned integer.
TrailingZeroCount(UIntPtr)	Computes the number of trailing zeros in a value.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.
TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormat Provider)	Tries to format the value of the current instance into the provided span of characters.

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UIntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UIntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Byte>, UIntPtr)	Tries to convert a UTF-8 character span containing the string representation of a number to its unsigned integer equivalent.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UIntPtr)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UIntPtr)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, UIntPtr)	Converts the read-only span of characters representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, IFormatProvider, UIntPtr)	Tries to parse a string into a value.
TryParse(String, NumberStyles, IFormatProvider, UIntPtr)	Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, UIntPtr)	Converts the string representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

Operators

Expand table

Addition(UIntPtr, Int32)	Adds an offset to an unsigned integer.
Equality(UIntPtr, UIntPtr)	Determines whether two specified instances of UIntPtr are equal.
Explicit(UInt32 to UIntPtr)	Converts the value of a 32-bit unsigned integer to an UIntPtr.
Explicit(UInt64 to UIntPtr)	Converts the value of a 64-bit unsigned integer to an UIntPtr.

Explicit(UIntPtr to UInt32)	Converts the value of the specified UIntPtr to a 32-bit unsigned integer.
Explicit(UIntPtr to UInt64)	Converts the value of the specified UIntPtr to a 64-bit unsigned integer.
Explicit(UIntPtr to Void*)	Converts the value of the specified UIntPtr to a pointer to an unspecified type. This API is not CLS-compliant.
Explicit(Void* to UIntPtr)	Converts the specified pointer to an unspecified type to an UIntPtr. This API is not CLS-compliant.
Inequality(UIntPtr, UIntPtr)	Determines whether two specified instances of UIntPtr are not equal.
Subtraction(UIntPtr, Int32)	Subtracts an offset from an unsigned integer.

Explicit Interface Implementations

Expand table

IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.Addition(UIntPtr, UIntPtr)	Adds two values together to compute their sum.
IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedAddition(UIntPtr, UIntPtr)	Adds two values together to compute their sum.
IAdditiveIdentity<UIntPtr,UIntPtr>.AdditiveIdentity	Gets the additive identity of the current type.
IBinaryInteger<UIntPtr>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<UIntPtr>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<UIntPtr>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UIntPtr)	
IBinaryInteger<UIntPtr>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UIntPtr)	
IBinaryInteger<UIntPtr>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<UIntPtr>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.

IBinaryNumber<UIntPtr>.AllBitsSet	Gets an instance of the binary type in which all bits are set.
IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseAnd(UIntPtr, UIntPtr)	Computes the bitwise-and of two values.
IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseOr(UIntPtr, UIntPtr)	Computes the bitwise-or of two values.
IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.ExclusiveOr(UIntPtr, UIntPtr)	Computes the exclusive-or of two values.
IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.OnesComplement(UIntPtr)	Computes the ones-complement representation of a given value.
IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThan(UIntPtr, UIntPtr)	Compares two values to determine which is greater.
IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThanOrEqual(UIntPtr, UIntPtr)	Compares two values to determine which is greater or equal.
IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThan(UIntPtr, UIntPtr)	Compares two values to determine which is less.
IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThanOrEqual(UIntPtr, UIntPtr)	Compares two values to determine which is less or equal.
IDecrementOperators<UIntPtr>.CheckedDecrement(UIntPtr)	Decrements a value.
IDecrementOperators<UIntPtr>.Decrement(UIntPtr)	Decrements a value.
IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>.Division(UIntPtr, UIntPtr)	Divides one value by another to compute their quotient.
IIncrementOperators<UIntPtr>.CheckedIncrement(UIntPtr)	Increments a value.
IIncrementOperators<UIntPtr>.Increment(UIntPtr)	Increments a value.
IMinMaxValue<UIntPtr>.MaxValue	Gets the maximum value of the current type.
IMinMaxValue<UIntPtr>.MinValue	Gets the minimum value of the current type.
IModulusOperators<UIntPtr,UIntPtr,UIntPtr>.Modulus(UIntPtr, UIntPtr)	Divides two values together to compute their modulus or remainder.

IMultiplicativeIdentity<UIntPtr,UIntPtr>.MultiplicativeIdentity	Gets the multiplicative identity of the current type.
IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedMultiply(UIntPtr, UIntPtr)	Multiplies two values together to compute their product.
IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.Multiply(UIntPtr, UIntPtr)	Multiplies two values together to compute their product.
INumber<UIntPtr>.CopySign(UIntPtr, UIntPtr)	Copies the sign of a value to the sign of another value.
INumber<UIntPtr>.MaxNumber(UIntPtr, UIntPtr)	Compares two values to compute which is greater and returning the other value if an input is `NaN`.
INumber<UIntPtr>.MinNumber(UIntPtr, UIntPtr)	Compares two values to compute which is lesser and returning the other value if an input is `NaN`.
INumberBase<UIntPtr>.Abs(UIntPtr)	Computes the absolute of a value.
INumberBase<UIntPtr>.IsCanonical(UIntPtr)	Determines if a value is in its canonical representation.
INumberBase<UIntPtr>.IsComplexNumber(UIntPtr)	Determines if a value represents a complex number.
INumberBase<UIntPtr>.IsFinite(UIntPtr)	Determines if a value is finite.
INumberBase<UIntPtr>.IsImaginaryNumber(UIntPtr)	Determines if a value represents a pure imaginary number.
INumberBase<UIntPtr>.IsInfinity(UIntPtr)	Determines if a value is infinite.
INumberBase<UIntPtr>.IsInteger(UIntPtr)	Determines if a value represents an integral number.
INumberBase<UIntPtr>.IsNaN(UIntPtr)	Determines if a value is NaN.
INumberBase<UIntPtr>.IsNegative(UIntPtr)	Determines if a value is negative.
INumberBase<UIntPtr>.IsNegativeInfinity(UIntPtr)	Determines if a value is negative infinity.
INumberBase<UIntPtr>.IsNormal(UIntPtr)	Determines if a value is normal.
INumberBase<UIntPtr>.IsPositive(UIntPtr)	Determines if a value is positive.
INumberBase<UIntPtr>.IsPositiveInfinity(UIntPtr)	Determines if a value is positive infinity.
INumberBase<UIntPtr>.IsRealNumber(UIntPtr)	Determines if a value represents a real number.

INumberBase<UIntPtr>.IsSubnormal(UIntPtr)	Determines if a value is subnormal.
INumberBase<UIntPtr>.IsZero(UIntPtr)	Determines if a value is zero.
INumberBase<UIntPtr>.MaxMagnitude(UIntPtr, UIntPtr)	Compares two values to compute which is greater.
INumberBase<UIntPtr>.MaxMagnitudeNumber(UIntPtr, UIntPtr)	Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.
INumberBase<UIntPtr>.MinMagnitude(UIntPtr, UIntPtr)	Compares two values to compute which is lesser.
INumberBase<UIntPtr>.MinMagnitudeNumber(UIntPtr, UIntPtr)	Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.
INumberBase<UIntPtr>.One	Gets the value `1` for the type.
INumberBase<UIntPtr>.Radix	Gets the radix, or base, for the type.
INumberBase<UIntPtr>.TryConvertFromChecked<TOther>(TOther, UIntPtr)	
INumberBase<UIntPtr>.TryConvertFromSaturating<TOther>(TOther, UIntPtr)	
INumberBase<UIntPtr>.TryConvertFromTruncating<TOther>(TOther, UIntPtr)	
INumberBase<UIntPtr>.TryConvertToChecked<TOther>(UIntPtr, TOther)	Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.
INumberBase<UIntPtr>.TryConvertToSaturating<TOther>(UIntPtr, TOther)	Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.
INumberBase<UIntPtr>.TryConvertToTruncating<TOther>(UIntPtr, TOther)	Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.
INumberBase<UIntPtr>.Zero	Gets the value `0` for the type.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data needed to serialize the current UIntPtr object.
IShiftOperators<UIntPtr,Int32,UIntPtr>.LeftShift(UIntPtr, Int32)	Shifts a value left by a given amount.
IShiftOperators<UIntPtr,Int32,UIntPtr>.RightShift(UIntPtr, Int32)	Shifts a value right by a given amount.

IShiftOperators<UIntPtr,Int32,UIntPtr>.UnsignedRightShift(UIntPtr, Int32)	Shifts a value right by a given amount.
ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedSubtraction(UIntPtr, UIntPtr)	Subtracts two values to compute their difference.
ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.Subtraction(UIntPtr, UIntPtr)	Subtracts two values to compute their difference.
IUnaryNegationOperators<UIntPtr,UIntPtr>.CheckedUnaryNegation(UIntPtr)	Computes the checked unary negation of a value.
IUnaryNegationOperators<UIntPtr,UIntPtr>.UnaryNegation(UIntPtr)	Computes the unary negation of a value.
IUnaryPlusOperators<UIntPtr,UIntPtr>.UnaryPlus(UIntPtr)	Computes the unary plus of a value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

This type is thread safe.

See also

- IntPtr

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of UIntPtr.

Overloads

Expand table

UIntPtr(UInt32)	Initializes a new instance of the UIntPtr structure using the specified 32-bit unsigned integer.
UIntPtr(UInt64)	Initializes a new instance of UIntPtr using the specified 64-bit unsigned integer.
UIntPtr(Void*)	Initializes a new instance of UIntPtr using the specified pointer to an unspecified type.

UIntPtr(UInt32)

Initializes a new instance of the UIntPtr structure using the specified 32-bit unsigned integer.

C#

```
public UIntPtr (uint value);
```

Parameters

`value` UInt32

A 32-bit unsigned integer.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UIntPtr(UInt64)

Initializes a new instance of UIntPtr using the specified 64-bit unsigned integer.

C#

```
public UIntPtr (ulong value);
```

Parameters

`value` UInt64

A 64-bit unsigned integer.

Exceptions

OverflowException

In a 32-bit process, `value` is too large to represent as an UIntPtr.

Remarks

An exception is only thrown if the value of `value` requires more bits than the executing process supports.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UIntPtr(Void*)

> Important
>
> This API is not CLS-compliant.

Initializes a new instance of UIntPtr using the specified pointer to an unspecified type.

```
C#
public UIntPtr (void* value);
```

Parameters

`value` Void*

A pointer to an unspecified type.

See also

- Void

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.Zero Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

A read-only field that represents an unsigned integer that has been initialized to zero.

```
C#
public static readonly UIntPtr Zero;
```

Field Value

UIntPtr

Remarks

The value of this field is not equivalent to `null`. Use this field to efficiently determine whether an instance of UIntPtr has been set to a value other than zero.

For example, assume the variable, uip, is an instance of UIntPtr. You can determine if it has been set by comparing it to the value returned by a constructor, for example, `if uip != new UIntPtr(0)...`. However, invoking a constructor to get an uninitialized pointer is inefficient. It is better to code either `if uip != UIntPtr.Zero...` or `if !UIntPtr.Zero.Equals(uip)...`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the largest possible value of UIntPtr.

C#

```
public static UIntPtr MaxValue { get; }
```

Property Value

UIntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the smallest possible value of UIntPtr.

C#

```
public static UIntPtr MinValue { get; }
```

Property Value

UIntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinValue

Applies to

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Size Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the size of this instance.

C#

```
public static int Size { get; }
```

Property Value

Int32

The size of an unsigned integer in the executing process, measured in bytes. The value of this property is 4 in a 32-bit process, and 8 in a 64-bit process. You can define the process type by setting the `/platform` switch when you compile your code with the C# and Visual Basic compilers.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr.Add(UIntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds an offset to an unsigned integer.

C#

```
public static UIntPtr Add (UIntPtr pointer, int offset);
```

Parameters

pointer UIntPtr

The unsigned integer to add the offset to.

offset Int32

The offset to add.

Returns

UIntPtr

A new unsigned integer that reflects the addition of `offset` to `pointer`.

Examples

The following example instantiates a UIntPtr object that points to the beginning of a ten-element array, and then calls the Add method to iterate the elements in the array.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
        UIntPtr ptr = (UIntPtr) arr[0];
        for (int ctr = 0; ctr < arr.Length; ctr++)
        {
           UIntPtr newPtr = UIntPtr.Add(ptr, ctr);
           Console.Write("{0}   ", newPtr);
        }
    }
}
// The example displays the following output:
//       1   2   3   4   5   6   7   8   9   10
```

Remarks

The Add method does not throw an exception if the result is too large to represent as an unsigned integer in the executing process. Instead, the addition operation is performed in an unchecked context.

Languages that do not support operator overloading or custom operators can use this method to add an offset to the value of a pointer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Addition(UIntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UIntPtr.Clamp(UIntPtr, UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Clamps a value to an inclusive minimum and maximum value.

C#

```
public static UIntPtr Clamp (UIntPtr value, UIntPtr min, UIntPtr max);
```

Parameters

`value` UIntPtr

The value to clamp.

`min` UIntPtr

The inclusive minimum to which `value` should clamp.

`max` UIntPtr

The inclusive maximum to which `value` should clamp.

Returns

UIntPtr

The result of clamping `value` to the inclusive range of `min` and `max`.

Implements

Clamp(TSelf, TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

CompareTo(Object)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.
CompareTo(UIntPtr)	Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

CompareTo(Object)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (object? value);
```

Parameters

`value` Object

An object to compare, or `null`.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

CompareTo(UIntPtr)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

C#

```
public int CompareTo (UIntPtr value);
```

Parameters

`value` UIntPtr

An unsigned native integer to compare.

Returns

Int32

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Expand table

Value	Meaning
Less than zero	This instance precedes `value` in the sort order.
Zero	This instance occurs in the same position in the sort order as `value`.
Greater than zero	This instance follows `value` in the sort order.

Implements

CompareTo(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.CreateChecked<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

public static UIntPtr CreateChecked<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value that's used to create the instance of `TSelf`.

Returns

UIntPtr

An instance of `TSelf` created from `value`.

Implements

CreateChecked<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.CreateSaturating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type.

```
C#

public static UIntPtr CreateSaturating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

UIntPtr

An instance of `TSelf` created from `value`, saturating if `value` falls outside the representable range of `TSelf`.

Implements

CreateSaturating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.CreateTruncating<TOther>(TOther) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type.

```
C#

public static UIntPtr CreateTruncating<TOther> (TOther value) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type of `value`.

Parameters

`value` TOther

The value which is used to create the instance of `TSelf`.

Returns

UIntPtr

An instance of `TSelf` created from `value`, truncating if `value` falls outside the representable range of `TSelf`.

Implements

CreateTruncating<TOther>(TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.DivRem(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the quotient and remainder of two values.

```
C#

public static (UIntPtr Quotient, UIntPtr Remainder) DivRem (UIntPtr left,
UIntPtr right);
```

Parameters

`left` UIntPtr

The value which `right` divides.

`right` UIntPtr

The value which divides `left`.

Returns

ValueTuple<UIntPtr,UIntPtr>

The quotient and remainder of `left` divided-by `right`.

Implements

DivRem(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Equals(Object)	Returns a value indicating whether this instance is equal to a specified object.
Equals(UIntPtr)	Indicates whether the current object is equal to another object of the same type.

Equals(Object)

Returns a value indicating whether this instance is equal to a specified object.

```
C#

public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with this instance or `null`.

Returns

Boolean

`true` if `obj` is an instance of UIntPtr and equals the value of this instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(UIntPtr)

Indicates whether the current object is equal to another object of the same type.

C#

```
public bool Equals (UIntPtr other);
```

Parameters

`other` UIntPtr

An object to compare with this object.

Returns

Boolean

`true` if the current object is equal to the `other` parameter; otherwise, `false`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.IsEvenInteger(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an even integral number.

```
C#
public static bool IsEvenInteger (UIntPtr value);
```

Parameters

value UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an even integer; otherwise, `false`.

Implements

IsEvenInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `2.0` will return `true` while `2.2` will return `false`.

A return value of `false` does not imply that IsOddInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IsOddInteger(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an odd integral number.

```
C#
public static bool IsOddInteger (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an odd integer; otherwise, `false`.

Implements

IsOddInteger(TSelf)

Remarks

This method correctly handles floating-point values and so `3.0` will return `true` while `3.3` will return `false`.

A return value of `false` does not imply that IsEvenInteger(TSelf) will return `true`. A number with a fractional portion, for example, `3.3`, is not even or odd.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IsPow2(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is a power of two.

C#

```
public static bool IsPow2 (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a power of two; otherwise, `false`.

Implements

IsPow2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.LeadingZeroCount(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of leading zeros in a value.

```
C#
public static UIntPtr LeadingZeroCount (UIntPtr value);
```

Parameters

`value` UIntPtr

The value whose leading zeroes are to be counted.

Returns

UIntPtr

The number of leading zeros in `value`.

Implements

LeadingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Log2(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the log2 of a value.

```
C#
public static UIntPtr Log2 (UIntPtr value);
```

Parameters

`value` UIntPtr

The value whose log2 is to be computed.

Returns

UIntPtr

The log2 of `value`.

Implements

Log2(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Max(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

```
C#
public static UIntPtr Max (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

Max(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `maximum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Min(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

```
C#
public static UIntPtr Min (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

Min(TSelf, TSelf)

Remarks

For IFloatingPoint<TSelf> this method matches the IEEE 754:2019 `minimum` function. This requires NaN inputs to be propagated back to the caller and for `-0.0` to be treated as less than `+0.0`.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(String)	Converts the string representation of a number to its unsigned native integer equivalent.
Parse(ReadOnlySpan<Byte>, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, IFormatProvider)	Parses a span of characters into a value.
Parse(String, NumberStyles)	Converts the string representation of a number in a specified style to its unsigned native integer equivalent.
Parse(String, IFormatProvider)	Converts the string representation of a number in a specified culture-specific format to its unsigned native integer equivalent.
Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)	Parses a span of UTF-8 characters into a value.
Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)	Converts the read-only span of characters representation of a number in optionally specified style and optionally specified culture-specific format to its unsigned native integer equivalent.
Parse(String, NumberStyles, IFormatProvider)	Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent.

Parse(String)

Converts the string representation of a number to its unsigned native integer equivalent.

```
C#
public static UIntPtr Parse (string s);
```

Parameters

s String

A string containing a number to convert.

Returns

UIntPtr

An unsigned native integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```
C#

public static UIntPtr Parse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UIntPtr

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, IFormatProvider)

Parses a span of characters into a value.

```
C#

public static UIntPtr Parse (ReadOnlySpan<char> s, IFormatProvider?
provider);
```

Parameters

`s` ReadOnlySpan<Char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

UIntPtr

The result of parsing `s`.

Implements

Parse(ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Parse(String, NumberStyles)

Converts the string representation of a number in a specified style to its unsigned native integer equivalent.

C#

```
public static UIntPtr Parse (string s, System.Globalization.NumberStyles
style);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`.

Returns

UIntPtr

An unsigned native integer equivalent to the number contained in `s`.

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(String, IFormatProvider)

Converts the string representation of a number in a specified culture-specific format to its unsigned native integer equivalent.

C#

```
public static UIntPtr Parse (string s, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UIntPtr

An unsigned native integer equivalent to the number contained in `s`.

Implements

Parse(String, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Parses a span of UTF-8 characters into a value.

```C#
public static UIntPtr Parse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

Returns

UIntPtr

The result of parsing `utf8Text`.

Implements

Parse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Converts the read-only span of characters representation of a number in optionally specified style and optionally specified culture-specific format to its unsigned native

integer equivalent.

```C#
public static UIntPtr Parse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style =
System.Globalization.NumberStyles.Integer, IFormatProvider? provider =
default);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`style` NumberStyles

An optional bitwise combination of the enumeration values that indicates the style elements that can be present in `s`. The default value is Integer.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information about `s`. The default value is `default`.

Returns

UIntPtr

An unsigned native integer equivalent to the number contained in `s`.

Implements

Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

Exceptions

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Parse(String, NumberStyles, IFormatProvider)

Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent.

```
C#

public static UIntPtr Parse (string s, System.Globalization.NumberStyles
style, IFormatProvider? provider);
```

Parameters

`s` String

A string containing a number to convert.

`style` NumberStyles

A bitwise combination of the enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

Returns

UIntPtr

An unsigned native integer equivalent to the number contained in `s`.

Implements

Parse(String, NumberStyles, IFormatProvider)

Exceptions

ArgumentNullException

`s` is `null`.

ArgumentException

`style` is not a NumberStyles value or `style` is not a combination of AllowHexSpecifier and HexNumber values.

FormatException

`s` is not in the correct format.

OverflowException

`s` represents a number less than MinValue or greater than MaxValue.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.PopCount(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of bits that are set in a value.

C#

```
public static UIntPtr PopCount (UIntPtr value);
```

Parameters

`value` UIntPtr

The value whose set bits are to be counted.

Returns

UIntPtr

The number of set bits in `value`.

Implements

PopCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.RotateLeft(UIntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value left by a given amount.

```
C#
public static UIntPtr RotateLeft (UIntPtr value, int rotateAmount);
```

Parameters

`value` UIntPtr

The value which is rotated left by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated left.

Returns

UIntPtr

The result of rotating `value` left by `rotateAmount`.

Implements

RotateLeft(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.RotateRight(UIntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Rotates a value right by a given amount.

```
C#
public static UIntPtr RotateRight (UIntPtr value, int rotateAmount);
```

Parameters

`value` UIntPtr

The value which is rotated right by `rotateAmount`.

`rotateAmount` Int32

The amount by which `value` is rotated right.

Returns

UIntPtr

The result of rotating `value` right by `rotateAmount`.

Implements

RotateRight(TSelf, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Sign(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the sign of a value.

```
C#
public static int Sign (UIntPtr value);
```

Parameters

`value` UIntPtr

The value whose sign is to be computed.

Returns

Int32

A positive value if `value` is positive, Zero if `value` is zero, and a negative value if `value` is negative.

Implements

Sign(TSelf)

Remarks

It is recommended that a function return `1`, `0`, and `-1`, respectively.

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Subtract(UIntPtr, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts an offset from an unsigned integer.

```
C#

public static UIntPtr Subtract (UIntPtr pointer, int offset);
```

Parameters

`pointer` UIntPtr

The unsigned integer to subtract the offset from.

`offset` Int32

The offset to subtract.

Returns

UIntPtr

A new unsigned integer that reflects the subtraction of `offset` from `pointer`.

Examples

The following example instantiates an IntPtr object that points to the end of a ten-element array, and then calls the Subtract method to iterate the elements in the array in reverse order.

```
C#

using System;

public class Example
{
   public static void Main()
```

```
    {
        int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        UIntPtr ptr = (UIntPtr) arr[arr.GetUpperBound(0)];
        for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
        {
            UIntPtr newPtr = UIntPtr.Subtract(ptr, ctr);
            Console.Write("{0}   ", newPtr);
        }
    }
}
// The example displays the following output:
//       10   9   8   7   6   5   4   3   2   1
```

Remarks

The Subtract method does not throw an exception if the result is too small to represent as an unsigned integer in the executing process. Instead, the subtraction operation is performed in an unchecked context.

Languages that do not support operator overloading or custom operators can use this method to subtract an offset from the value of a pointer.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtraction(UIntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr.ToPointer Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Converts the value of this instance to a pointer to an unspecified type.

```
C#
public void* ToPointer ();
```

Returns

Void*

A pointer to Void; that is, a pointer to memory containing data of an unspecified type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

ToString()	Converts the numeric value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
ToString(String)	Converts the numeric value of this instance to its equivalent string representation, using the specified format.
ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.

ToString()

Converts the numeric value of this instance to its equivalent string representation.

C#

```
public override string ToString ();
```

Returns

String

The string representation of the value of this instance.

Remarks

If the value of the Size property for this instance is 4, then this method is equivalent to ToString; otherwise, this method is equivalent to ToString.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(IFormatProvider)

Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.

C#

```
public string ToString (IFormatProvider? provider);
```

Parameters

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The string representation of the value of this instance as specified by `provider`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(String)

Converts the numeric value of this instance to its equivalent string representation, using the specified format.

```
C#
public string ToString (string? format);
```

Parameters

`format` String

A standard or custom numeric format string.

Returns

String

The string representation of the value of this instance as specified by `format`.

Exceptions

FormatException

`format` is invalid or not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

ToString(String, IFormatProvider)

Formats the value of the current instance using the specified format.

```
C#
public string ToString (string? format, IFormatProvider? provider);
```

Parameters

`format` String

The format to use. -or- A `null` reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`provider` IFormatProvider

An object that supplies culture-specific formatting information.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Exceptions

FormatException

`format` is invalid or not supported.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr.ToUInt32 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to a 32-bit unsigned integer.

```
C#
public uint ToUInt32 ();
```

Returns

UInt32

A 32-bit unsigned integer equal to the value of this instance.

Exceptions

OverflowException

In a 64-bit process, the value of this instance is too large to represent as a 32-bit unsigned integer.

Remarks

An exception is only thrown if the value of `value` requires more bits than the executing process supports.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.ToUInt64 Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of this instance to a 64-bit unsigned integer.

C#

```
public ulong ToUInt64 ();
```

Returns

UInt64

A 64-bit unsigned integer equal to the value of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.TrailingZeroCount(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the number of trailing zeros in a value.

```
C#
public static UIntPtr TrailingZeroCount (UIntPtr value);
```

Parameters

`value` UIntPtr

The value whose trailing zeroes are to be counted.

Returns

UIntPtr

The number of trailing zeros in `value`.

Implements

TrailingZeroCount(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.
TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance into the provided span of characters.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten,
ReadOnlySpan<char> format = default, IFormatProvider? provider =
default);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UIntPtr.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(String, IFormatProvider, UIntPtr)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Char>, UIntPtr)	Converts the read-only span of characters representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(String, UIntPtr)	Converts the string representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Byte>, IFormatProvider, UIntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, IFormatProvider, UIntPtr)	Tries to parse a string into a value.
TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UIntPtr)	Tries to parse a span of UTF-8 characters into a value.
TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UIntPtr)	Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Byte>, UIntPtr)	Tries to convert a UTF-8 character span containing the string representation of a number to its unsigned integer equivalent.
TryParse(String, NumberStyles, IFormatProvider, UIntPtr)	Converts the string representation of a number in a specified style and culture-specific format to its

	unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

TryParse(String, IFormatProvider, UIntPtr)

Tries to parse a string into a value.

```
C#

public static bool TryParse (string? s, IFormatProvider? provider, out
UIntPtr result);
```

Parameters

`s` String

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UIntPtr

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Char>, UIntPtr)

Converts the read-only span of characters representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> s, out UIntPtr result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`result` UIntPtr

When this method returns, contains the unsigned native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(String, UIntPtr)

Converts the string representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

```
C#
```

```
public static bool TryParse (string? s, out UIntPtr result);
```

Parameters

`s` String

A string containing a number to convert.

`result` UIntPtr

When this method returns, contains the unsigned native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

TryParse(ReadOnlySpan<Byte>, IFormatProvider, UIntPtr)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
IFormatProvider? provider, out UIntPtr result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UIntPtr

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, IFormatProvider, UIntPtr)

Tries to parse a string into a value.

C#

```
public static bool TryParse (ReadOnlySpan<char> s, IFormatProvider?
provider, out UIntPtr result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` UIntPtr

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

TryParse(ReadOnlySpan<Byte>, NumberStyles, IFormatProvider, UIntPtr)

Tries to parse a span of UTF-8 characters into a value.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
UIntPtr result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

The span of UTF-8 characters to parse.

`style` NumberStyles

A bitwise combination of number styles that can be present in `utf8Text`.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `utf8Text`.

`result` UIntPtr

On return, contains the result of successfully parsing `utf8Text` or an undefined value on failure.

Returns

Boolean

`true` if `utf8Text` was successfully parsed; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, UIntPtr)

Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse (ReadOnlySpan<char> s,
System.Globalization.NumberStyles style, IFormatProvider? provider, out
UIntPtr result);
```

Parameters

`s` ReadOnlySpan<Char>

A read-only span of characters containing a number to convert. The span is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UIntPtr

When this method returns, contains the unsigned native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryParse(ReadOnlySpan<Byte>, UIntPtr)

Tries to convert a UTF-8 character span containing the string representation of a number to its unsigned integer equivalent.

C#

```
public static bool TryParse (ReadOnlySpan<byte> utf8Text, out UIntPtr
result);
```

Parameters

`utf8Text` ReadOnlySpan<Byte>

A span containing the UTF-8 characters representing the number to convert.

`result` UIntPtr

When this method returns, contains the unsigned integer value equivalent to the number contained in `utf8Text` if the conversion succeeded, or zero if the conversion failed. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `utf8Text` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryParse(String, NumberStyles, IFormatProvider, UIntPtr)

Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? s, System.Globalization.NumberStyles
style, IFormatProvider? provider, out UIntPtr result);
```

Parameters

`s` String

A string containing a number to convert. The string is interpreted using the style specified by `style`.

`style` NumberStyles

A bitwise combination of enumeration values that indicates the style elements that can be present in `s`.

`provider` IFormatProvider

An object that supplies culture-specific formatting information about `s`.

`result` UIntPtr

When this method returns, contains the unsigned native integer value equivalent of the number contained in `s`, if the conversion succeeded, or zero if the conversion failed. The conversion fails if the `s` parameter is `null` or empty, is not of the correct format, or represents a number less than MinValue or greater than MaxValue. This parameter is passed uninitialized; any value originally supplied in result will be overwritten.

Returns

Boolean

`true` if `s` was converted successfully; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	5, 6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.Addition(UIntPtr, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds an offset to an unsigned integer.

C#

```
public static UIntPtr operator + (UIntPtr pointer, int offset);
```

Parameters

`pointer` UIntPtr

The unsigned integer to add the offset to.

`offset` Int32

The offset to add.

Returns

UIntPtr

A new unsigned integer that reflects the addition of `offset` to `pointer`.

Remarks

The Addition method defines the addition operation for UIntPtr objects. It enables code such as the following.

C#

```
int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
UIntPtr ptr = (UIntPtr) arr[0];
for (int ctr = 0; ctr < arr.Length; ctr++)
{
```

```
    UIntPtr newPtr = ptr + ctr;
    Console.WriteLine(newPtr);
}
```

Languages that do not support custom operators can call the Add method instead.

The addition operation does not throw an exception if the result is too large to represent as an unsigned integer in the executing process. Instead, it is performed in an unchecked context.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, this API is only accessible via reflection. The addition operator is directly recognized by the language and will follow the normal language behavior for addition operations, including overflowing in a `checked` context if the result is too large to represent.

The equivalent method for this operator is UIntPtr.Add(UIntPtr, Int32).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Add(UIntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.Equality(UIntPtr, UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of UIntPtr are equal.

```
C#
public static bool operator == (UIntPtr value1, UIntPtr value2);
```

Parameters

`value1` UIntPtr

The first unsigned integer to compare.

`value2` UIntPtr

The second unsigned integer to compare.

Returns

Boolean

`true` if `value1` equals `value2`; otherwise, `false`.

Implements

Equality(TSelf, TOther)

Remarks

The equivalent method for this operator is UIntPtr.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.Explicit Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Explicit(UInt32 to UIntPtr)	Converts the value of a 32-bit unsigned integer to an UIntPtr.
Explicit(UInt64 to UIntPtr)	Converts the value of a 64-bit unsigned integer to an UIntPtr.
Explicit(UIntPtr to UInt32)	Converts the value of the specified UIntPtr to a 32-bit unsigned integer.
Explicit(UIntPtr to UInt64)	Converts the value of the specified UIntPtr to a 64-bit unsigned integer.
Explicit(UIntPtr to Void*)	Converts the value of the specified UIntPtr to a pointer to an unspecified type. This API is not CLS-compliant.
Explicit(Void* to UIntPtr)	Converts the specified pointer to an unspecified type to an UIntPtr. This API is not CLS-compliant.

Explicit(UInt32 to UIntPtr)

Converts the value of a 32-bit unsigned integer to an UIntPtr.

C#

```
public static explicit operator UIntPtr (uint value);
```

Parameters

`value` UInt32

A 32-bit unsigned integer.

Returns

UIntPtr

A new instance of UIntPtr initialized to `value`.

See also

- UIntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(UInt64 to UIntPtr)

Converts the value of a 64-bit unsigned integer to an UIntPtr.

C#

```
public static explicit operator UIntPtr (ulong value);
```

Parameters

`value` UInt64

A 64-bit unsigned integer.

Returns

UIntPtr

A new instance of UIntPtr initialized to `value`.

Exceptions

OverflowException

In a 32-bit process, `value` is too large to represent as an UIntPtr.

See also

- UIntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(UIntPtr to UInt32)

Converts the value of the specified UIntPtr to a 32-bit unsigned integer.

C#

```
public static explicit operator uint (UIntPtr value);
```

Parameters

`value` UIntPtr

The unsigned integer to convert.

Returns

UInt32

The contents of `value`.

Exceptions

OverflowException

In a 64-bit process, the value of `value` is too large to represent as a 32-bit unsigned integer.

Remarks

An exception is only thrown if the value of `value` requires more bits than the executing process supports.

See also

- ToUInt32()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(UIntPtr to UInt64)

Converts the value of the specified UIntPtr to a 64-bit unsigned integer.

```
C#
```

```
public static explicit operator ulong (UIntPtr value);
```

Parameters

`value` UIntPtr

The unsigned integer to convert.

Returns

UInt64

The contents of `value`.

See also

- ToUInt64()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(UIntPtr to Void*)

> Important
>
> This API is not CLS-compliant.

Converts the value of the specified UIntPtr to a pointer to an unspecified type.

This API is not CLS-compliant.

```C#
public static explicit operator void* (UIntPtr value);
```

Parameters

`value` UIntPtr

The unsigned integer to convert.

Returns

Void*

The contents of `value`.

See also

- ToPointer()

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Explicit(Void* to UIntPtr)

> Important
>
> This API is not CLS-compliant.

Converts the specified pointer to an unspecified type to an UIntPtr.

This API is not CLS-compliant.

```
C#

public static explicit operator UIntPtr (void* value);
```

Parameters

`value` Void*

A pointer to an unspecified type.

Returns

UIntPtr

A new instance of UIntPtr initialized to `value`.

See also

- UIntPtr

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.Inequality(UIntPtr, UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified instances of UIntPtr are not equal.

```
C#
public static bool operator != (UIntPtr value1, UIntPtr value2);
```

Parameters

`value1` UIntPtr

The first unsigned integer to compare.

`value2` UIntPtr

The second unsigned integer to compare.

Returns

Boolean

`true` if `value1` does not equal `value2`; otherwise, `false`.

Implements

Inequality(TSelf, TOther)

Remarks

The equivalent method for this operator is UIntPtr.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.Subtraction(UIntPtr, Int32) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts an offset from an unsigned integer.

```
C#

public static UIntPtr operator - (UIntPtr pointer, int offset);
```

Parameters

`pointer` UIntPtr

The unsigned integer to subtract the offset from.

`offset` Int32

The offset to subtract.

Returns

UIntPtr

A new unsigned integer that reflects the subtraction of `offset` from `pointer`.

Remarks

The Subtraction method defines the subtraction operation for UIntPtr objects. It enables code such as the following.

```
C#

int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
UIntPtr ptr = (UIntPtr) arr[arr.GetUpperBound(0)];
for (int ctr = 0; ctr <= arr.GetUpperBound(0); ctr++)
{
```

```
    UIntPtr newPtr = ptr - ctr;
    Console.Write("{0}   ", newPtr);
}
```

Languages that do not support custom operators can call the Subtract method instead.

The subtraction operation does not throw an exception if the result is too small to represent as an unsigned integer in the executing process. Instead, it is performed in an unchecked context.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, this API is only accessible via reflection. The subtraction operator is directly recognized by the language and will follow the normal language behavior for subtraction operations, including overflowing in a `checked` context if the result is too small to represent.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Subtract(UIntPtr, Int32)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.Addition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

```
C#

static UIntPtr IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.operator +
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to which `right` is added.

`right` UIntPtr

The value that's added to `left`.

Returns

UIntPtr

The sum of `left` and `right`.

Implements

Addition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedAddition Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Adds two values together to compute their sum.

C#

```
static UIntPtr
IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.op_CheckedAddition (UIntPtr
left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to which `right` is added.

`right` UIntPtr

The value that's added to `left`.

Returns

UIntPtr

The sum of `left` and `right`.

Implements

CheckedAddition(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IAdditiveIdentity<UIntPtr,UIntPtr>.AdditiveIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the additive identity of the current type.

C#

```
static UIntPtr
System.Numerics.IAdditiveIdentity<System.UIntPtr,System.UIntPtr>.AdditiveIde
ntity { get; }
```

Property Value

UIntPtr

Implements

AdditiveIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.GetByteCount Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

C#

```
int IBinaryInteger<UIntPtr>.GetByteCount ();
```

Returns

Int32

The number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).

Implements

GetByteCount()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.GetShortestBitLength Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length, in bits, of the shortest two's complement representation of the current value.

C#

```
int IBinaryInteger<UIntPtr>.GetShortestBitLength ();
```

Returns

Int32

The length, in bits, of the shortest two's complement representation of the current value.

Implements

GetShortestBitLength()

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.TryReadBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<UIntPtr>.TryReadBigEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out UIntPtr value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UIntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.TryReadLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool IBinaryInteger<UIntPtr>.TryReadLittleEndian (ReadOnlySpan<byte>
source, bool isUnsigned, out UIntPtr value);
```

Parameters

`source` ReadOnlySpan<Byte>

`isUnsigned` Boolean

`value` UIntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.TryWriteBigEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in big-endian format, to a given span.

C#

```
bool IBinaryInteger<UIntPtr>.TryWriteBigEndian (Span<byte> destination, out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteBigEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IBinaryInteger<UIntPtr>.TryWriteLittleEndian Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to write the current value, in little-endian format, to a given span.

C#

```
bool IBinaryInteger<UIntPtr>.TryWriteLittleEndian (Span<byte> destination,
out int bytesWritten);
```

Parameters

destination Span<Byte>

The span to which the current value should be written.

bytesWritten Int32

When this method returns, contains the number of bytes written to `destination`.

Returns

Boolean

`true` if the value was succesfully written to `destination`; otherwise, `false`.

Implements

TryWriteLittleEndian(Span<Byte>, Int32)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IBinaryNumber<UIntPtr>.AllBitsSet Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an instance of the binary type in which all bits are set.

C#

```
static UIntPtr System.Numerics.IBinaryNumber<System.UIntPtr>.AllBitsSet {
get; }
```

Property Value

UIntPtr

Implements

AllBitsSet

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseAnd Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-and of two values.

C#

```
static UIntPtr IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.operator &
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to `and` with `right`.

`right` UIntPtr

The value to `and` with `left`.

Returns

UIntPtr

The bitwise-and of `left` and `right`.

Implements

BitwiseAnd(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the bitwise-or of two values.

C#

```
static UIntPtr IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.operator |
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to `or` with `right`.

`right` UIntPtr

The value to `or` with `left`.

Returns

UIntPtr

The bitwise-or of `left` and `right`.

Implements

BitwiseOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.ExclusiveOr Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the exclusive-or of two values.

```
C#

static UIntPtr IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.operator ^
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to xor with `right`.

`right` UIntPtr

The value to xor with `left`.

Returns

UIntPtr

The exclusive-or of `left` and `right`.

Implements

ExclusiveOr(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.OnesComplement Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the ones-complement representation of a given value.

C#

```
static UIntPtr IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.operator ~
(UIntPtr value);
```

Parameters

`value` UIntPtr

The value for which to compute the ones-complement.

Returns

UIntPtr

The ones-complement of `value`.

Implements

OnesComplement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThanOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater.

```
C#

static bool IComparisonOperators<UIntPtr,UIntPtr,bool>.operator > (UIntPtr
left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to compare with `right`.

`right` UIntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than `right`; otherwise, `false`.

Implements

GreaterThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThanOrEqual Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is greater or equal.

```
C#

static bool IComparisonOperators<UIntPtr,UIntPtr,bool>.operator >= (UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to compare with `right`.

`right` UIntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is greater than or equal to `right`; otherwise, `false`.

Implements

GreaterThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThan Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less.

```
C#

static bool IComparisonOperators<UIntPtr,UIntPtr,bool>.operator < (UIntPtr
left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to compare with `right`.

`right` UIntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than `right`; otherwise, `false`.

Implements

LessThan(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThanOrEqualOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to determine which is less or equal.

```
C#

static bool IComparisonOperators<UIntPtr,UIntPtr,bool>.operator <= (UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value to compare with `right`.

`right` UIntPtr

The value to compare with `left`.

Returns

Boolean

`true` if `left` is less than or equal to `right`; otherwise, `false`.

Implements

LessThanOrEqual(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IDecrementOperators<UIntPtr>.CheckedDecrement(UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

```
C#

static UIntPtr IDecrementOperators<UIntPtr>.op_CheckedDecrement (UIntPtr
value);
```

Parameters

`value` UIntPtr

The value to decrement.

Returns

UIntPtr

The result of decrementing `value`.

Implements

CheckedDecrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IDecrementOperators<UIntPtr>.Decrement(UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Decrements a value.

C#

```
static UIntPtr IDecrementOperators<UIntPtr>.operator -- (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to decrement.

Returns

UIntPtr

The result of decrementing `value`.

Implements

Decrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>.Division Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides one value by another to compute their quotient.

```
C#

static UIntPtr IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>.operator /
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value that `right` divides.

`right` UIntPtr

The value that divides `left`.

Returns

UIntPtr

The quotient of `left` divided by `right`.

Implements

Division(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IIncrementOperators<UIntPtr>.CheckedIncrement(UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#

static UIntPtr IIncrementOperators<UIntPtr>.op_CheckedIncrement (UIntPtr
value);
```

Parameters

`value` UIntPtr

The value to increment.

Returns

UIntPtr

The result of incrementing `value`.

Implements

CheckedIncrement(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IIncrementOperators<UIntPtr>.Increment(UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Increments a value.

```
C#
static UIntPtr IIncrementOperators<UIntPtr>.operator ++ (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to increment.

Returns

UIntPtr

The result of incrementing `value`.

Implements

Increment(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IMinMaxValue<UIntPtr>.MaxValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the maximum value of the current type.

C#

```
static UIntPtr System.Numerics.IMinMaxValue<System.UIntPtr>.MaxValue { get;
}
```

Property Value

UIntPtr

Implements

MaxValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.IMinMaxValue<UIntPtr>.MinValue Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the minimum value of the current type.

C#

```
static UIntPtr System.Numerics.IMinMaxValue<System.UIntPtr>.MinValue { get;
}
```

Property Value

UIntPtr

Implements

MinValue

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.IModulusOperators<UIntPtr,UIntPtr,UIntPtr>.Modulus Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Divides two values together to compute their modulus or remainder.

```
C#

static UIntPtr IModulusOperators<UIntPtr,UIntPtr,UIntPtr>.operator %
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value that `right` divides.

`right` UIntPtr

The value that divides `left`.

Returns

UIntPtr

The modulus or remainder of `left` divided by `right`.

Implements

Modulus(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IMultiplicativeIdentity<UIntPtr,UIntPtr>.MultiplicativeIdentity Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the multiplicative identity of the current type.

C#

```
static UIntPtr
System.Numerics.IMultiplicativeIdentity<System.UIntPtr,System.UIntPtr>.Multi
plicativeIdentity { get; }
```

Property Value

UIntPtr

Implements

MultiplicativeIdentity

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UIntPtr.IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedMultiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

C#

```
static UIntPtr
IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.op_CheckedMultiply (UIntPtr
left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value that `right` multiplies.

`right` UIntPtr

The value that multiplies `left`.

Returns

UIntPtr

The product of `left` multiplied by `right`.

Implements

CheckedMultiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.Multiply Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Multiplies two values together to compute their product.

```
C#

static UIntPtr IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.operator *
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value that `right` multiplies.

`right` UIntPtr

The value that multiplies `left`.

Returns

UIntPtr

The product of `left` multiplied by `right`.

Implements

Multiply(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumber<UIntPtr>.CopySign(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Copies the sign of a value to the sign of another value.

C#

```
static UIntPtr INumber<UIntPtr>.CopySign (UIntPtr value, UIntPtr sign);
```

Parameters

`value` UIntPtr

The value whose magnitude is used in the result.

`sign` UIntPtr

The value whose sign is used in the result.

Returns

UIntPtr

A value with the magnitude of `value` and the sign of `sign`.

Implements

CopySign(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumber<UIntPtr>.MaxNumber(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater and returning the other value if an input is `NaN`.

```
C#
static UIntPtr INumber<UIntPtr>.MaxNumber (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumber<UIntPtr>.MinNumber(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser and returning the other value if an input is `NaN`.

```
C#
static UIntPtr INumber<UIntPtr>.MinNumber (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.Abs(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the absolute of a value.

```
C#
static UIntPtr INumberBase<UIntPtr>.Abs (UIntPtr value);
```

Parameters

`value` UIntPtr

The value for which to get its absolute.

Returns

UIntPtr

The absolute of `value`.

Implements

Abs(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsCanonical(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is in its canonical representation.

C#

```
static bool INumberBase<UIntPtr>.IsCanonical (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is in its canonical representation; otherwise, `false`.

Implements

IsCanonical(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsComplexNumber(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a complex number.

C#

```
static bool INumberBase<UIntPtr>.IsComplexNumber (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a complex number; otherwise, `false`.

Implements

IsComplexNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsFinite(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is finite.

```
C#
static bool INumberBase<UIntPtr>.IsFinite (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is finite; otherwise, `false`.

Implements

IsFinite(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsImaginaryNumber(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a pure imaginary number.

C#

```
static bool INumberBase<UIntPtr>.IsImaginaryNumber (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a pure imaginary number; otherwise, `false`.

Implements

IsImaginaryNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsInfinity(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is infinite.

```
C#
static bool INumberBase<UIntPtr>.IsInfinity (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is infinite; otherwise, `false`.

Implements

IsInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsInteger(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents an integral number.

```
C#
static bool INumberBase<UIntPtr>.IsInteger (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is an integer; otherwise, `false`.

Implements

IsInteger(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsNaN(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is NaN.

```
C#
static bool INumberBase<UIntPtr>.IsNaN (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is NaN; otherwise, `false`.

Implements

IsNaN(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsNegative(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative.

```
C#
static bool INumberBase<UIntPtr>.IsNegative (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is negative; otherwise, `false`.

Implements

IsNegative(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsNegativeInfinity(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is negative infinity.

C#

```
static bool INumberBase<UIntPtr>.IsNegativeInfinity (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is negative infinity; otherwise, `false`.

Implements

IsNegativeInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsNormal(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is normal.

```
C#
static bool INumberBase<UIntPtr>.IsNormal (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is normal; otherwise, `false`.

Implements

IsNormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsPositive(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive.

C#

```
static bool INumberBase<UIntPtr>.IsPositive (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is positive; otherwise, `false`.

Implements

IsPositive(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsPositiveInfinity(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is positive infinity.

C#

```
static bool INumberBase<UIntPtr>.IsPositiveInfinity (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is positive infinity; otherwise, `false`.

Implements

IsPositiveInfinity(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsRealNumber(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value represents a real number.

C#

```
static bool INumberBase<UIntPtr>.IsRealNumber (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is a real number; otherwise, `false`.

Implements

IsRealNumber(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsSubnormal(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is subnormal.

```
C#
static bool INumberBase<UIntPtr>.IsSubnormal (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is subnormal; otherwise, `false`.

Implements

IsSubnormal(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.IsZero(UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines if a value is zero.

```
C#
static bool INumberBase<UIntPtr>.IsZero (UIntPtr value);
```

Parameters

`value` UIntPtr

The value to be checked.

Returns

Boolean

`true` if `value` is zero; otherwise, `false`.

Implements

IsZero(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.MaxMagnitude(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is greater.

C#

```
static UIntPtr INumberBase<UIntPtr>.MaxMagnitude (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.MaxMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`.

C#

```
static UIntPtr INumberBase<UIntPtr>.MaxMagnitudeNumber (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is greater than `y`; otherwise, `y`.

Implements

MaxMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.MinMagnitude(UIntPtr, UIntPtr) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which is lesser.

C#

```
static UIntPtr INumberBase<UIntPtr>.MinMagnitude (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitude(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.MinMagnitudeNumber Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`.

C#

```
static UIntPtr INumberBase<UIntPtr>.MinMagnitudeNumber (UIntPtr x, UIntPtr y);
```

Parameters

x UIntPtr

The value to compare with `y`.

y UIntPtr

The value to compare with `x`.

Returns

UIntPtr

`x` if it is less than `y`; otherwise, `y`.

Implements

MinMagnitudeNumber(TSelf, TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.One Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `1` for the type.

C#

```
static UIntPtr System.Numerics.INumberBase<System.UIntPtr>.One { get; }
```

Property Value

UIntPtr

Implements

One

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.INumberBase<UIntPtr>.Radix Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the radix, or base, for the type.

C#

```
static int System.Numerics.INumberBase<System.UIntPtr>.Radix { get; }
```

Property Value

Int32

Implements

Radix

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertFromChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UIntPtr>.TryConvertFromChecked<TOther> (TOther
value, out UIntPtr result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UIntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertFromSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UIntPtr>.TryConvertFromSaturating<TOther> (TOther
value, out UIntPtr result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UIntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertFromTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

C#

```
static bool INumberBase<UIntPtr>.TryConvertFromTruncating<TOther> (TOther
value, out UIntPtr result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

Parameters

`value` TOther

`result` UIntPtr

Returns

Boolean

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertToChecked<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<UIntPtr>.TryConvertToChecked<TOther> (UIntPtr value,
out TOther result) where TOther : System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UIntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToChecked<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertToSaturating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<UIntPtr>.TryConvertToSaturating<TOther> (UIntPtr
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UIntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToSaturating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.TryConvertToTruncating<TOther> Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type.

```
C#

static bool INumberBase<UIntPtr>.TryConvertToTruncating<TOther> (UIntPtr
value, out TOther result) where TOther :
System.Numerics.INumberBase<TOther>;
```

Type Parameters

`TOther`

The type to which `value` should be converted.

Parameters

`value` UIntPtr

The value that's used to create the instance of `TOther`.

`result` TOther

When this method returns, contains an instance of `TOther` converted from `value`.

Returns

Boolean

`false` if `TOther` is not supported; otherwise, `true`.

Implements

TryConvertToTruncating<TOther>(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.INumberBase<UIntPtr>.Zero Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value `0` for the type.

C#

```
static UIntPtr System.Numerics.INumberBase<System.UIntPtr>.Zero { get; }
```

Property Value

UIntPtr

Implements

Zero

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UIntPtr.IShiftOperators<UIntPtr,Int32,UIntPtr>.LeftShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value left by a given amount.

C#

```
static UIntPtr IShiftOperators<UIntPtr,int,UIntPtr>.operator << (UIntPtr
value, int shiftAmount);
```

Parameters

value UIntPtr

The value that is shifted left by `shiftAmount`.

shiftAmount Int32

The amount by which `value` is shifted left.

Returns

UIntPtr

The result of shifting `value` left by `shiftAmount`.

Implements

LeftShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IShiftOperators<UIntPtr,Int32,UIntPtr>.RightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

C#

```
static UIntPtr IShiftOperators<UIntPtr,int,UIntPtr>.operator >> (UIntPtr
value, int shiftAmount);
```

Parameters

value UIntPtr

The value that is shifted right by `shiftAmount`.

shiftAmount Int32

The amount by which `value` is shifted right.

Returns

UIntPtr

The result of shifting `value` right by `shiftAmount`.

Implements

RightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IShiftOperators<UIntPtr,Int32,UIntPtr>.UnsignedRightShift Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Shifts a value right by a given amount.

```
C#

static UIntPtr IShiftOperators<UIntPtr,int,UIntPtr>.op_UnsignedRightShift
(UIntPtr value, int shiftAmount);
```

Parameters

`value` UIntPtr

The value that is shifted right by `shiftAmount`.

`shiftAmount` Int32

The amount by which `value` is shifted right.

Returns

UIntPtr

The result of shifting `value` right by `shiftAmount`.

Implements

UnsignedRightShift(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedSubtractionOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

C#

```
static UIntPtr
ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.op_CheckedSubtraction
(UIntPtr left, UIntPtr right);
```

Parameters

left UIntPtr

The value from which `right` is subtracted.

right UIntPtr

The value that's subtracted from `left`.

Returns

UIntPtr

The value of `right` subtracted from `left`.

Implements

CheckedSubtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.Subtraction Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Subtracts two values to compute their difference.

```
C#

static UIntPtr ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.operator -
(UIntPtr left, UIntPtr right);
```

Parameters

`left` UIntPtr

The value from which `right` is subtracted.

`right` UIntPtr

The value that's subtracted from `left`.

Returns

UIntPtr

The value of `right` subtracted from `left`.

Implements

Subtraction(TSelf, TOther)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IUnaryNegationOperators<UIntPtr,UIntPtr>.CheckedUnaryNegationOperator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the checked unary negation of a value.

```
C#

static UIntPtr
IUnaryNegationOperators<UIntPtr,UIntPtr>.op_CheckedUnaryNegation (UIntPtr
value);
```

Parameters

`value` UIntPtr

The value for which to compute the unary negation.

Returns

UIntPtr

The unary negation of `value`.

Implements

CheckedUnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IUnaryNegationOperators<UIntPtr,UIntPtr>.UnaryNegation Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary negation of a value.

```
C#

static UIntPtr IUnaryNegationOperators<UIntPtr,UIntPtr>.operator - (UIntPtr
value);
```

Parameters

`value` UIntPtr

The value for which to compute the unary negation.

Returns

UIntPtr

The unary negation of `value`.

Implements

UnaryNegation(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.IUnaryPlusOperators<UIntPtr,UIntPtr>.UnaryPlus(UIntPtr) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Computes the unary plus of a value.

```
C#

static UIntPtr IUnaryPlusOperators<UIntPtr,UIntPtr>.operator + (UIntPtr
value);
```

Parameters

`value` UIntPtr

The value for which to compute the unary plus.

Returns

UIntPtr

The unary plus of `value`.

Implements

UnaryPlus(TSelf)

Applies to

Product	Versions
.NET	7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UIntPtr.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo object with the data needed to serialize the current UIntPtr object.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo object to populate with data.

`context` StreamingContext

The destination for this serialization. (This parameter is not used; specify `null`.)

Implements

GetObjectData(SerializationInfo, StreamingContext)

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

This method populates the `info` parameter with the value of the current UIntPtr object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UnauthorizedAccessException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when the operating system denies access because of an I/O error or a specific type of security error.

C#

```
public class UnauthorizedAccessException : SystemException
```

Inheritance Object → Exception → SystemException → UnauthorizedAccessException

Derived System.Security.AccessControl.PrivilegeNotHeldException

Examples

The following example illustrates the UnauthorizedAccessException exception that is thrown when attempting to write to a read-only file.

C#

```
using System;
using System.IO;

public class Example
{
   public static void Main()
   {
      string filePath = @".\ROFile.txt";
      if (! File.Exists(filePath))
         File.Create(filePath);
      // Keep existing attributes, and set ReadOnly attribute.
      File.SetAttributes(filePath,
                         (new FileInfo(filePath)).Attributes |
FileAttributes.ReadOnly);

      StreamWriter sw = null;
      try {
         sw = new StreamWriter(filePath);
```

```
            sw.Write("Test");
        }
        catch (UnauthorizedAccessException) {
            FileAttributes attr = (new FileInfo(filePath)).Attributes;
            Console.Write("UnAuthorizedAccessException: Unable to access file.
");
            if ((attr & FileAttributes.ReadOnly) > 0)
                Console.Write("The file is read-only.");
         }
         finally {
            if (sw != null) sw.Close();
        }
    }
}
// The example displays the following output:
//    UnAuthorizedAccessException: Unable to access file. The file is read-
only.
```

Remarks

An UnauthorizedAccessException exception is typically thrown by a method that wraps a Windows API call. To find the reasons for the exception, examine the text of the exception object's Message property.

UnauthorizedAccessException uses the `HRESULT` `COR_E_UNAUTHORIZEDACCESS`, which has the value 0x80070005.

Constructors

Expand table

UnauthorizedAccessException()	Initializes a new instance of the UnauthorizedAccessException class.
UnauthorizedAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the UnauthorizedAccessException class with serialized data.
UnauthorizedAccessException(String)	Initializes a new instance of the UnauthorizedAccessException class with a specified error message.
UnauthorizedAccessException(String, Exception)	Initializes a new instance of the UnauthorizedAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.**

	When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Exception
- Handling and Throwing Exceptions

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UnauthorizedAccessException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the UnauthorizedAccessException class.

Overloads

Expand table

UnauthorizedAccessException()	Initializes a new instance of the UnauthorizedAccessException class.
UnauthorizedAccessException(String)	Initializes a new instance of the UnauthorizedAccessException class with a specified error message.
UnauthorizedAccessException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the UnauthorizedAccessException class with serialized data.
UnauthorizedAccessException(String, Exception)	Initializes a new instance of the UnauthorizedAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

UnauthorizedAccessException()

Initializes a new instance of the UnauthorizedAccessException class.

C#

```
public UnauthorizedAccessException ();
```

Remarks

This constructor initializes the Message property of the new instance to a system-supplied message that describes the error, such as "Attempted to perform an unauthorized operation." This message takes into account the current system culture.

The following table shows the initial property values for an instance of MemberAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The localized error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UnauthorizedAccessException(String)

Initializes a new instance of the UnauthorizedAccessException class with a specified error message.

C#

```
public UnauthorizedAccessException (string? message);
```

Parameters

`message` String

The message that describes the error.

Remarks

The content of the `message` parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

The following table shows the initial property values for an instance of UnauthorizedAccessException.

Expand table

Property	Value
InnerException	A null reference (`Nothing` in Visual Basic).
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UnauthorizedAccessException(SerializationInfo, StreamingContext)

⊗ **Caution**

This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the UnauthorizedAccessException class with serialized data.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected UnauthorizedAccessException
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

The SerializationInfo that holds the serialized object data about the exception being thrown.

`context` StreamingContext

The StreamingContext that contains contextual information about the source or destination.

Attributes ObsoleteAttribute

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

UnauthorizedAccessException(String, Exception)

Initializes a new instance of the UnauthorizedAccessException class with a specified error message and a reference to the inner exception that is the cause of this exception.

```
C#

public UnauthorizedAccessException (string? message, Exception? inner);
```

Parameters

`message` String

The error message that explains the reason for the exception.

`inner` Exception

The exception that is the cause of the current exception. If the `inner` parameter is not a null reference (`Nothing` in Visual Basic), the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or a null reference (`Nothing` in Visual Basic) if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of UnauthorizedAccessException.

Expand table

Property	Value
InnerException	The inner exception reference.
Message	The error message string.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UnhandledExceptionEventArgs Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides data for the event that is raised when there is an exception that is not handled in any application domain.

C#

```
public class UnhandledExceptionEventArgs : EventArgs
```

Inheritance Object → EventArgs → UnhandledExceptionEventArgs

Remarks

UnhandledExceptionEventArgs provides access to the exception object and a flag indicating whether the common language runtime is terminating. The UnhandledExceptionEventArgs is one of the parameters passed into UnhandledExceptionEventHandler for the AppDomain.UnhandledException event.

For more information about event handler delegates, see Handling and Raising Events.

Constructors

Expand table

UnhandledExceptionEventArgs(Object, Boolean)	Initializes a new instance of the UnhandledExceptionEventArgs class with the exception object and a common language runtime termination flag.

Properties

Expand table

Exception Object	Gets the unhandled exception object.
IsTerminating	Gets a value that indicates whether the common language runtime is terminating.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- UnhandledException
- UnhandledExceptionEventHandler

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UnhandledExceptionEventArgs(Object, Boolean) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the UnhandledExceptionEventArgs class with the exception object and a common language runtime termination flag.

C#

```
public UnhandledExceptionEventArgs (object exception, bool isTerminating);
```

Parameters

`exception` Object

The exception that is not handled.

`isTerminating` Boolean

`true` if the runtime is terminating; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UnhandledExceptionEventArgs.ExceptionObject Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the unhandled exception object.

C#

```
public object ExceptionObject { get; }
```

Property Value

Object

The unhandled exception object.

Examples

The following example demonstrates the UnhandledException event. It defines an event handler, `MyHandler`, that is invoked whenever an unhandled exception is thrown in the default application domain. It then throws two exceptions. The first is handled by a **try/catch** block. The second is unhandled and invokes the `MyHandle` routine before the application terminates.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      AppDomain currentDomain = AppDomain.CurrentDomain;
      currentDomain.UnhandledException += new
UnhandledExceptionEventHandler(MyHandler);

      try {
```

```
            throw new Exception("1");
        } catch (Exception e) {
            Console.WriteLine("Catch clause caught : {0} \n", e.Message);
        }

        throw new Exception("2");
    }

    static void MyHandler(object sender, UnhandledExceptionEventArgs args)
    {
        Exception e = (Exception) args.ExceptionObject;
        Console.WriteLine("MyHandler caught : " + e.Message);
        Console.WriteLine("Runtime terminating: {0}", args.IsTerminating);
    }
}
// The example displays the following output:
//       Catch clause caught : 1
//
//       MyHandler caught : 2
//       Runtime terminating: True
//
//       Unhandled Exception: System.Exception: 2
//          at Example.Main()
```

Remarks

This property returns an object of type Object rather than one derived from Exception. Although the Common Language Specification requires that all exception types derive from Exception, it is possible for methods to throw exceptions with objects not derived from Exception. You can do the following to work with this exception:

- Apply the RuntimeCompatibilityAttribute attribute with a RuntimeCompatibilityAttribute.WrapNonExceptionThrows value of `true` to the assembly that contains the event handler. This wraps all exceptions not derived from the Exception class in a RuntimeWrappedException object. You can then safely cast (in C#) or convert (in Visual Basic) the object returned by this property to an Exception object, and retrieve the original exception object from the RuntimeWrappedException.WrappedException property. Note that some compilers, such as the C# and Visual Basic compilers, automatically apply this attribute.

- Cast the object returned by this property to an Exception object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- RuntimeCompatibilityAttribute
- RuntimeWrappedException

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UnhandledExceptionEventArgs.IsTerminating Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the common language runtime is terminating.

```
C#
public bool IsTerminating { get; }
```

Property Value

Boolean

`true` if the runtime is terminating; otherwise, `false`.

Remarks

This property returns `true` for most unhandled exceptions. The reason is that most unhandled exceptions in threads, including thread pool threads, are allowed to proceed naturally, which normally results in termination of the application. For more information, including a list of cases in which the runtime does not terminate, see Exceptions in Managed Threads.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UnhandledExceptionEventHandler Delegate

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the method that will handle the event raised by an exception that is not handled by the application domain.

C#

```
public delegate void UnhandledExceptionEventHandler(object sender,
UnhandledExceptionEventArgs e);
```

Parameters

sender Object

The source of the unhandled exception event.

e UnhandledExceptionEventArgs

An `UnhandledExceptionEventArgs` that contains the event data.

Remarks

An UnhandledExceptionEventHandler can only be specified for the default application domain that is created by the system to execute an application. Specifying an UnhandledExceptionEventHandler for an AppDomain created by an application has no effect.

When you create an UnhandledExceptionEventHandler delegate, you identify the method that will handle the event. To associate the event handler with your event, add an instance of the delegate to the event. The event handler is called whenever the event occurs, unless you remove the delegate. For more information about event handler delegates, see Handling and Raising Events.

Extension Methods

Expand table

GetMethodInfo(Delegate)	Gets an object that represents the method represented by the specified delegate.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- UnhandledExceptionEventArgs
- EventHandler

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI.

```C#
public class Uri : ISpanFormattable,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → Uri

Implements ISerializable , IFormattable , ISpanFormattable

Examples

The following example creates an instance of the Uri class and uses it to perform a GET request with HttpClient.

```C#
Uri siteUri = new Uri("http://www.contoso.com/");

// HttpClient lifecycle management best practices:
//
https://learn.microsoft.com/dotnet/fundamentals/networking/http/httpclient-
guidelines#recommended-use
HttpClient client = new HttpClient();
HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get,
siteUri);
HttpResponseMessage response = client.Send(request);
```

The following code snippet shows example values of the various properties on the class.

```C#
```

```
Uri uri = new Uri("https://user:password@www.contoso.com:80/Home/Index.htm?
q1=v1&q2=v2#FragmentName");

Console.WriteLine($"AbsolutePath: {uri.AbsolutePath}");
Console.WriteLine($"AbsoluteUri: {uri.AbsoluteUri}");
Console.WriteLine($"DnsSafeHost: {uri.DnsSafeHost}");
Console.WriteLine($"Fragment: {uri.Fragment}");
Console.WriteLine($"Host: {uri.Host}");
Console.WriteLine($"HostNameType: {uri.HostNameType}");
Console.WriteLine($"IdnHost: {uri.IdnHost}");
Console.WriteLine($"IsAbsoluteUri: {uri.IsAbsoluteUri}");
Console.WriteLine($"IsDefaultPort: {uri.IsDefaultPort}");
Console.WriteLine($"IsFile: {uri.IsFile}");
Console.WriteLine($"IsLoopback: {uri.IsLoopback}");
Console.WriteLine($"IsUnc: {uri.IsUnc}");
Console.WriteLine($"LocalPath: {uri.LocalPath}");
Console.WriteLine($"OriginalString: {uri.OriginalString}");
Console.WriteLine($"PathAndQuery: {uri.PathAndQuery}");
Console.WriteLine($"Port: {uri.Port}");
Console.WriteLine($"Query: {uri.Query}");
Console.WriteLine($"Scheme: {uri.Scheme}");
Console.WriteLine($"Segments: {string.Join(", ", uri.Segments)}");
Console.WriteLine($"UserEscaped: {uri.UserEscaped}");
Console.WriteLine($"UserInfo: {uri.UserInfo}");

// AbsolutePath: /Home/Index.htm
// AbsoluteUri: https://user:password@www.contoso.com:80/Home/Index.htm?
q1=v1&q2=v2#FragmentName
// DnsSafeHost: www.contoso.com
// Fragment: #FragmentName
// Host: www.contoso.com
// HostNameType: Dns
// IdnHost: www.contoso.com
// IsAbsoluteUri: True
// IsDefaultPort: False
// IsFile: False
// IsLoopback: False
// IsUnc: False
// LocalPath: /Home/Index.htm
// OriginalString: https://user:password@www.contoso.com:80/Home/Index.htm?
q1=v1&q2=v2#FragmentName
// PathAndQuery: /Home/Index.htm?q1=v1&q2=v2
// Port: 80
// Query: ?q1=v1&q2=v2
// Scheme: https
// Segments: /, Home/, Index.htm
// UserEscaped: False
// UserInfo: user:password
```

Remarks

For more information about this API, see Supplemental API remarks for Uri.

Constructors

Expand table

Uri(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the Uri class from the specified instances of the SerializationInfo and StreamingContext classes.
Uri(String)	Initializes a new instance of the Uri class with the specified URI.
Uri(String, Boolean)	**Obsolete.** Initializes a new instance of the Uri class with the specified URI, with explicit control of character escaping.
Uri(String, UriCreationOptions)	Initializes a new instance of the Uri class with the specified URI and additional UriCreationOptions.
Uri(String, UriKind)	Initializes a new instance of the Uri class with the specified URI. This constructor allows you to specify if the URI string is a relative URI, absolute URI, or is indeterminate.
Uri(Uri, String)	Initializes a new instance of the Uri class based on the specified base URI and relative URI string.
Uri(Uri, String, Boolean)	**Obsolete.** Initializes a new instance of the Uri class based on the specified base and relative URIs, with explicit control of character escaping.
Uri(Uri, Uri)	Initializes a new instance of the Uri class based on the combination of a specified base Uri instance and a relative Uri instance.

Fields

Expand table

SchemeDelimiter	Specifies the characters that separate the communication protocol scheme from the address portion of the URI. This field is read-only.
UriSchemeFile	Specifies that the URI is a pointer to a file. This field is read-only.
UriSchemeFtp	Specifies that the URI is accessed through the File Transfer Protocol (FTP). This field is read-only.

UriScheme Ftps	Specifies that the URI is accessed through the File Transfer Protocol Secure (FTPS). This field is read-only.
UriScheme Gopher	Specifies that the URI is accessed through the Gopher protocol. This field is read-only.
UriScheme Http	Specifies that the URI is accessed through the Hypertext Transfer Protocol (HTTP). This field is read-only.
UriScheme Https	Specifies that the URI is accessed through the Secure Hypertext Transfer Protocol (HTTPS). This field is read-only.
UriScheme Mailto	Specifies that the URI is an email address and is accessed through the Simple Mail Transport Protocol (SMTP). This field is read-only.
UriSchemeNet Pipe	Specifies that the URI is accessed through the NetPipe scheme used by Windows Communication Foundation (WCF). This field is read-only.
UriSchemeNet Tcp	Specifies that the URI is accessed through the NetTcp scheme used by Windows Communication Foundation (WCF). This field is read-only.
UriScheme News	Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.
UriScheme Nntp	Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.
UriScheme Sftp	Specifies that the URI is accessed through the SSH File Transfer Protocol (SFTP). This field is read-only.
UriSchemeSsh	Specifies that the URI is accessed through the Secure Socket Shell protocol (SSH). This field is read-only.
UriScheme Telnet	Specifies that the URI is accessed through the Telnet protocol. This field is read-only.
UriSchemeWs	Specifies that the URI is accessed through the WebSocket protocol (WS). This field is read-only.
UriScheme Wss	Specifies that the URI is accessed through the WebSocket Secure protocol (WSS). This field is read-only.

Properties

Expand table

Absolute Path	Gets the absolute path of the URI.

AbsoluteUri	Gets the absolute URI.
Authority	Gets the Domain Name System (DNS) host name or IP address and the port number for a server.
DnsSafe Host	Gets a host name that, after being unescaped if necessary, is safe to use for DNS resolution.
Fragment	Gets the escaped URI fragment, including the leading '#' character if not empty.
Host	Gets the host component of this instance.
HostName Type	Gets the type of the host name specified in the URI.
IdnHost	Gets the RFC 3490 compliant International Domain Name of the host, using Punycode as appropriate. This string, after being unescaped if necessary, is safe to use for DNS resolution.
IsAbsolute Uri	Gets a value that indicates whether the Uri instance is absolute.
IsDefault Port	Gets a value that indicates whether the port value of the URI is the default for this scheme.
IsFile	Gets a value that indicates whether the specified Uri is a file URI.
IsLoopback	Gets a value that indicates whether the specified Uri references the local host.
IsUnc	Gets a value that indicates whether the specified Uri is a universal naming convention (UNC) path.
LocalPath	Gets a local operating-system representation of a file name.
Original String	Gets the original URI string that was passed to the Uri constructor.
PathAnd Query	Gets the AbsolutePath and Query properties separated by a question mark (?).
Port	Gets the port number of this URI.
Query	Gets any query information included in the specified URI, including the leading '?' character if not empty.
Scheme	Gets the scheme name for this URI.
Segments	Gets an array containing the path segments that make up the specified URI.
User Escaped	Gets a value that indicates whether the URI string was completely escaped before the Uri instance was created.

UserInfo	Gets the user name, password, or other user-specific information associated with the specified URI.

Methods

Expand table

Canonicalize()	**Obsolete.** Converts the internally stored URI to canonical form.
CheckHostName(String)	Determines whether the specified host name is a valid DNS name.
CheckSchemeName(String)	Determines whether the specified scheme name is valid.
CheckSecurity()	**Obsolete.** Calling this method has no effect.
Compare(Uri, Uri, UriComponents, UriFormat, StringComparison)	Compares the specified parts of two URIs using the specified comparison rules.
Equals(Object)	Compares two Uri instances for equality.
Escape()	**Obsolete.** Converts any unsafe or reserved characters in the path component to their hexadecimal character representations.
EscapeDataString(String)	Converts a string to its escaped representation.
EscapeString(String)	**Obsolete.** Converts a string to its escaped representation.
EscapeUriString(String)	**Obsolete.** Converts a URI string to its escaped representation.
FromHex(Char)	Gets the decimal value of a hexadecimal digit.
GetComponents(UriComponents, UriFormat)	Gets the specified components of the current instance using the specified escaping for special characters.
GetHashCode()	Gets the hash code for the URI.
GetLeftPart(UriPartial)	Gets the specified portion of a Uri instance.

GetObjectData(SerializationInfo, StreamingContext)	Returns the data needed to serialize the current instance.
GetType()	Gets the Type of the current instance. (Inherited from Object)
HexEscape(Char)	Converts a specified character into its hexadecimal equivalent.
HexUnescape(String, Int32)	Converts a specified hexadecimal representation of a character to the character.
IsBadFileSystemCharacter(Char)	**Obsolete.** Indicates whether a character is invalid in a file system name.
IsBaseOf(Uri)	Determines whether the current Uri instance is a base of the specified Uri instance.
IsExcludedCharacter(Char)	**Obsolete.** Determines whether the specified character should be escaped.
IsHexDigit(Char)	Determines whether a specified character is a valid hexadecimal digit.
IsHexEncoding(String, Int32)	Determines whether a character in a string is hexadecimal encoded.
IsReservedCharacter(Char)	**Obsolete.** Determines whether the specified character is a reserved character.
IsWellFormedOriginalString()	Indicates whether the string used to construct this Uri was well-formed and does not require further escaping.
IsWellFormedUriString(String, UriKind)	Indicates whether the string is well-formed by attempting to construct a URI with the string and ensures that the string does not require further escaping.
MakeRelative(Uri)	**Obsolete.** Determines the difference between two Uri instances.
MakeRelativeUri(Uri)	Determines the difference between two Uri instances.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Parse()	**Obsolete.**

	Parses the URI of the current instance to ensure it contains all the parts required for a valid URI.
ToString()	Gets a canonical string representation for the specified Uri instance.
TryCreate(String, UriCreationOptions, Uri)	Creates a new Uri using the specified String instance and UriCreationOptions.
TryCreate(String, UriKind, Uri)	Creates a new Uri using the specified String instance and a UriKind.
TryCreate(Uri, String, Uri)	Creates a new Uri using the specified base and relative String instances.
TryCreate(Uri, Uri, Uri)	Creates a new Uri using the specified base and relative Uri instances.
TryFormat(Span<Char>, Int32)	Attempts to format a canonical string representation for the Uri instance into the specified span.
Unescape(String)	**Obsolete.** Converts the specified string by replacing any escape sequences with their unescaped representation.
UnescapeDataString(String)	Converts a string to its unescaped representation.

Operators

Expand table

Equality(Uri, Uri)	Determines whether two Uri instances have the same value.
Inequality(Uri, Uri)	Determines whether two Uri instances do not have the same value.

Explicit Interface Implementations

Expand table

IFormattable.ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Returns the data needed to serialize the current instance.

ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Thread Safety

All members of Uri are thread-safe and may be used concurrently from multiple threads.

See also

- IdnElement
- IriParsingElement
- UriSection
- DnsSafeHost
- MakeRelative(Uri)
- IsWellFormedOriginalString()
- UriBuilder
- Changes to the System.Uri namespace in Version 2.0
- International Resource Identifier Support in System.UriSystem.Uri
- Network Programming in the .NET Framework

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Uri Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Uri class.

Overloads

Expand table

Uri(String)	Initializes a new instance of the Uri class with the specified URI.
Uri(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the Uri class from the specified instances of the SerializationInfo and StreamingContext classes.
Uri(String, Boolean)	**Obsolete.** Initializes a new instance of the Uri class with the specified URI, with explicit control of character escaping.
Uri(String, UriCreationOptions)	Initializes a new instance of the Uri class with the specified URI and additional UriCreationOptions.
Uri(String, UriKind)	Initializes a new instance of the Uri class with the specified URI. This constructor allows you to specify if the URI string is a relative URI, absolute URI, or is indeterminate.
Uri(Uri, String)	Initializes a new instance of the Uri class based on the specified base URI and relative URI string.
Uri(Uri, Uri)	Initializes a new instance of the Uri class based on the combination of a specified base Uri instance and a relative Uri instance.
Uri(Uri, String, Boolean)	**Obsolete.** Initializes a new instance of the Uri class based on the specified base and relative URIs, with explicit control of character escaping.

Uri(String)

Initializes a new instance of the Uri class with the specified URI.

```
C#

public Uri (string uriString);
```

Parameters

`uriString` String

A string that identifies the resource to be represented by the Uri instance. Note that an IPv6 address in string form must be enclosed within brackets. For example, "http://[2607:f8b0:400d:c06::69]".

Exceptions

ArgumentNullException

`uriString` is `null`.

UriFormatException

Note: In the .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

`uriString` is empty.

-or-

The scheme specified in `uriString` is not correctly formed. See CheckSchemeName(String).

-or-

`uriString` contains too many slashes.

-or-

The password specified in `uriString` is not valid.

-or-

The host name specified in `uriString` is not valid.

-or-

The file name specified in `uriString` is not valid.

-or-

The user name specified in `uriString` is not valid.

-or-

The host or authority name specified in `uriString` cannot be terminated by backslashes.

-or-

The port number specified in `uriString` is not valid or cannot be parsed.

-or-

The length of `uriString` exceeds 65519 characters.

-or-

The length of the scheme specified in `uriString` exceeds 1023 characters.

-or-

There is an invalid character sequence in `uriString`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Examples

The following example creates a Uri instance with the URI `http://www.contoso.com/`.

```
C#

Uri myUri = new Uri("http://www.contoso.com/");
```

Remarks

This constructor creates a Uri instance from a URI string. It parses the URI, puts it in canonical format, and makes any required escape encodings.

This constructor does not ensure that the Uri refers to an accessible resource.

This constructor assumes that the `string` parameter references an absolute URI and is equivalent to calling the Uri constructor with UriKind set to Absolute. If the `string` parameter passed to the constructor is a relative URI, this constructor will throw a UriFormatException.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Uri(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the Uri class from the specified instances of the SerializationInfo and StreamingContext classes.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected Uri (System.Runtime.Serialization.SerializationInfo
serializationInfo, System.Runtime.Serialization.StreamingContext
streamingContext);
```

Parameters

`serializationInfo` SerializationInfo

An instance of the SerializationInfo class containing the information required to serialize the new Uri instance.

`streamingContext` StreamingContext

An instance of the StreamingContext class containing the source of the serialized stream associated with the new Uri instance.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

The `serializationInfo` parameter contains a `null` URI.

UriFormatException

The `serializationInfo` parameter contains a URI that is empty.

-or-

The scheme specified is not correctly formed. See CheckSchemeName(String).

-or-

The URI contains too many slashes.

-or-

The password specified in the URI is not valid.

-or-

The host name specified in URI is not valid.

-or-

The file name specified in the URI is not valid.

-or-

The user name specified in the URI is not valid.

-or-

The host or authority name specified in the URI cannot be terminated by backslashes.

-or-

The port number specified in the URI is not valid or cannot be parsed.

-or-

The length of URI exceeds 65519 characters.

-or-

The length of the scheme specified in the URI exceeds 1023 characters.

-or-

There is an invalid character sequence in the URI.

-or-

The MS-DOS path specified in the URI must start with c:\\.

Remarks

This constructor implements the ISerializable interface for the Uri class.

See also

- XML and SOAP Serialization

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Uri(String, Boolean)

⊗ **Caution**

This constructor has been deprecated; the dontEscape parameter is always false. Use Uri(string) instead.

Initializes a new instance of the Uri class with the specified URI, with explicit control of character escaping.

C#

```
[System.Obsolete("This constructor has been deprecated; the dontEscape parameter is always false. Use Uri(string) instead.")]
public Uri (string uriString, bool dontEscape);
```

Parameters

`uriString` String

A string that identifies the resource to be represented by the Uri instance. Note that an IPv6 address in string form must be enclosed within brackets. For example, "http://[2607:f8b0:400d:c06::69]".

`dontEscape` Boolean

`true` if `uriString` is completely escaped; otherwise, `false`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`uriString` is `null`.

UriFormatException

`uriString` is empty or contains only spaces.

-or-

The scheme specified in `uriString` is not valid.

-or-

`uriString` contains too many slashes.

-or-

The password specified in `uriString` is not valid.

-or-

The host name specified in `uriString` is not valid.

-or-

The file name specified in `uriString` is not valid.

-or-

The user name specified in `uriString` is not valid.

-or-

The host or authority name specified in `uriString` cannot be terminated by backslashes.

-or-

The port number specified in `uriString` is not valid or cannot be parsed.

-or-

The length of `uriString` exceeds 65519 characters.

-or-

The length of the scheme specified in `uriString` exceeds 1023 characters.

-or-

There is an invalid character sequence in `uriString`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Examples

The following example creates a Uri instance for the URI `http://www.contoso.com/Hello%20World.htm`. Because the contained URI is completely escaped and is in canonical form, the `dontEscape` parameter can be set to `true`.

```
C#
```

```
Uri myUri = new Uri("http://www.contoso.com/Hello%20World.htm", true);
```

Remarks

This constructor creates a Uri instance from a URI string. It parses the URI and puts it in canonical format.

The `dontEscape` parameter controls whether reserved characters are translated into escape sequences. This parameter should be set to `true` only if you are certain that all reserved characters in the URI have been escaped. Setting the value to `true` for a URI that has not been completely escaped can cause unexpected behavior. It is strongly recommended that you always set this parameter to `false`.

If `dontEscape` is set to `false`, the constructor escapes any reserved characters by checking that all occurrences of percent (%) are followed by a valid escape sequence. If the character sequence following a percent is not valid, the percent is replaced by %25.

This constructor does not ensure that the Uri refers to an accessible resource.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Uri(String, UriCreationOptions)

Initializes a new instance of the Uri class with the specified URI and additional UriCreationOptions.

```
C#
```

```
public Uri (string uriString, in UriCreationOptions creationOptions);
```

Parameters

`uriString` String

A string that identifies the resource to be represented by the Uri instance.

`creationOptions` UriCreationOptions

Options that control how the Uri is created and behaves.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Uri(String, UriKind)

Initializes a new instance of the Uri class with the specified URI. This constructor allows you to specify if the URI string is a relative URI, absolute URI, or is indeterminate.

C#

```
public Uri (string uriString, UriKind uriKind);
```

Parameters

`uriString` String

A string that identifies the resource to be represented by the Uri instance. Note that an IPv6 address in string form must be enclosed within brackets. For example, "http://[2607:f8b0:400d:c06::69]".

`uriKind` UriKind

Specifies whether the URI string is a relative URI, absolute URI, or is indeterminate.

Exceptions

ArgumentException

`uriKind` is invalid.

ArgumentNullException

`uriString` is `null`.

UriFormatException

Note: In the .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

`uriString` contains a relative URI and `uriKind` is Absolute.

or

`uriString` contains an absolute URI and `uriKind` is Relative.

or

`uriString` is empty.

-or-

The scheme specified in `uriString` is not correctly formed. See CheckSchemeName(String).

-or-

`uriString` contains too many slashes.

-or-

The password specified in `uriString` is not valid.

-or-

The host name specified in `uriString` is not valid.

-or-

The file name specified in `uriString` is not valid.

-or-

The user name specified in `uriString` is not valid.

-or-

The host or authority name specified in `uriString` cannot be terminated by backslashes.

-or-

The port number specified in `uriString` is not valid or cannot be parsed.

-or-

The length of `uriString` exceeds 65519 characters.

-or-

The length of the scheme specified in `uriString` exceeds 1023 characters.

-or-

There is an invalid character sequence in `uriString`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Remarks

Relative and absolute URIs have different restrictions on their format. For example, a relative URI does not require a scheme or an authority. The value you specify in `uriKind` must match the type of URI passed in `uriString`. However, if RelativeOrAbsolute is specified, the URI string can be relative or absolute.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Uri(Uri, String)

Initializes a new instance of the Uri class based on the specified base URI and relative URI string.

```
C#

public Uri (Uri baseUri, string? relativeUri);
```

Parameters

`baseUri` Uri

The base URI.

`relativeUri` String

The relative URI to add to the base URI.

Exceptions

ArgumentNullException

`baseUri` is `null`.

ArgumentOutOfRangeException

`baseUri` is not an absolute Uri instance.

UriFormatException

Note: In the .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

The URI formed by combining `baseUri` and `relativeUri` is empty or contains only spaces.

-or-

The scheme specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The URI formed by combining `baseUri` and `relativeUri` contains too many slashes.

-or-

The password specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The file name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The user name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host or authority name specified in the URI formed by combining `baseUri` and `relativeUri` cannot be terminated by backslashes.

-or-

The port number specified in the URI formed by combining `baseUri` and `relativeUri` is not valid or cannot be parsed.

-or-

The length of the URI formed by combining `baseUri` and `relativeUri` exceeds 65519 characters.

-or-

The length of the scheme specified in the URI formed by combining `baseUri` and `relativeUri` exceeds 1023 characters.

-or-

There is an invalid character sequence in the URI formed by combining `baseUri` and `relativeUri`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Examples

The following example creates a new instance of the Uri class by combining the relative URIs `http://www.contoso.com` and `catalog/shownew.htm` to form the absolute URI `http://www.contoso.com/catalog/shownew.htm`.

```
C#

Uri baseUri = new Uri("http://www.contoso.com");
 Uri myUri = new Uri(baseUri, "catalog/shownew.htm");

Console.WriteLine(myUri.ToString());
```

Remarks

This constructor creates a Uri instance by combining the `baseUri` and the `relativeUri`. If `relativeUri` is an absolute URI (containing a scheme, host name, and optionally a port number), the Uri instance is created using only `relativeUri`.

If the `baseUri` has relative parts (like `/api`), then the relative part must be terminated with a slash, (like `/api/`), if the relative part of `baseUri` is to be preserved in the constructed Uri.

Additionally, if the `relativeUri` begins with a slash, then it will replace any relative part of the `baseUri`

This constructor does not ensure that the Uri refers to an accessible resource.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Uri(Uri, Uri)

Initializes a new instance of the Uri class based on the combination of a specified base Uri instance and a relative Uri instance.

```
C#

public Uri (Uri baseUri, Uri relativeUri);
```

Parameters

`baseUri` Uri

An absolute Uri that is the base for the new Uri instance.

`relativeUri` Uri

A relative Uri instance that is combined with `baseUri`.

Exceptions

ArgumentException

`baseUri` is not an absolute Uri instance.

ArgumentNullException

`baseUri` is `null`.

ArgumentOutOfRangeException

`baseUri` is not an absolute Uri instance.

UriFormatException

Note: In the .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

The URI formed by combining `baseUri` and `relativeUri` is empty or contains only spaces.

-or-

The scheme specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The URI formed by combining `baseUri` and `relativeUri` contains too many slashes.

-or-

The password specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The file name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The user name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host or authority name specified in the URI formed by combining `baseUri` and `relativeUri` cannot be terminated by backslashes.

-or-

The port number specified in the URI formed by combining `baseUri` and `relativeUri` is not valid or cannot be parsed.

-or-

The length of the URI formed by combining `baseUri` and `relativeUri` exceeds 65519 characters.

-or-

The length of the scheme specified in the URI formed by combining `baseUri` and `relativeUri` exceeds 1023 characters.

-or-

There is an invalid character sequence in the URI formed by combining `baseUri` and `relativeUri`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Examples

This example creates an absolute Uri instance, `absoluteUri`, and a relative Uri instance, `relativeUri`. A new Uri instance, `combinedUri`, is then created from these two instances.

C#

```
// Create an absolute Uri from a string.
Uri absoluteUri = new Uri("http://www.contoso.com/");

// Create a relative Uri from a string.  allowRelative = true to allow
for
// creating a relative Uri.
Uri relativeUri = new Uri("/catalog/shownew.htm?date=today",
UriKind.Relative);

// Check whether the new Uri is absolute or relative.
if (!relativeUri.IsAbsoluteUri)
    Console.WriteLine("{0} is a relative Uri.", relativeUri);

// Create a new Uri from an absolute Uri and a relative Uri.
Uri combinedUri = new Uri(absoluteUri, relativeUri);
Console.WriteLine(combinedUri.AbsoluteUri);
```

Remarks

This constructor creates a new Uri instance by combining an absolute Uri instance, `baseUri`, with a relative Uri instance, `relativeUri`. If `relativeUri` is an absolute Uri instance (containing a scheme, host name, and optionally a port number), the Uri instance is created using only `relativeUri`.

If the `baseUri` has relative parts (like `/api`), then the relative part must be terminated with a slash, (like `/api/`), if the relative part of `baseUri` is to be preserved in the constructed Uri.

Additionally, if the `relativeUri` begins with a slash, then it will replace any relative part of the `baseUri`

This constructor does not ensure that the Uri refers to an accessible resource.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Uri(Uri, String, Boolean)

> ⊗ **Caution**
>
> This constructor has been deprecated; the dontEscape parameter is always false. Use Uri(Uri, string) instead.

Initializes a new instance of the Uri class based on the specified base and relative URIs, with explicit control of character escaping.

C#

```
[System.Obsolete("This constructor has been deprecated; the dontEscape parameter is always false. Use Uri(Uri, string) instead.")]
public Uri (Uri baseUri, string? relativeUri, bool dontEscape);
```

Parameters

`baseUri` Uri

The base URI.

`relativeUri` String

The relative URI to add to the base URI.

`dontEscape` Boolean

`true` if `uriString` is completely escaped; otherwise, `false`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`baseUri` is `null`.

ArgumentOutOfRangeException

`baseUri` is not an absolute Uri instance.

UriFormatException

The URI formed by combining `baseUri` and `relativeUri` is empty or contains only spaces.

-or-

The scheme specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The URI formed by combining `baseUri` and `relativeUri` contains too many slashes.

-or-

The password specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The file name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The user name specified in the URI formed by combining `baseUri` and `relativeUri` is not valid.

-or-

The host or authority name specified in the URI formed by combining `baseUri` and `relativeUri` cannot be terminated by backslashes.

-or-

The port number specified in the URI formed by combining `baseUri` and `relativeUri` is not valid or cannot be parsed.

-or-

The length of the URI formed by combining `baseUri` and `relativeUri` exceeds 65519 characters.

-or-

The length of the scheme specified in the URI formed by combining `baseUri` and `relativeUri` exceeds 1023 characters.

-or-

There is an invalid character sequence in the URI formed by combining `baseUri` and `relativeUri`.

-or-

The MS-DOS path specified in `uriString` must start with c:\\.

Examples

The following example creates a new instance of the Uri class by combining the relative URIs `http://www.contoso.com` and `Hello%20World.htm` to form an absolute URI.

```
C#

Uri baseUri = new Uri("http://www.contoso.com");
Uri myUri = new Uri(baseUri, "Hello%20World.htm",false);
```

Remarks

This constructor creates a Uri instance by combining `baseUri` and `relativeUri`. If the URI passed in `relativeUri` is an absolute URI (containing a scheme, host name, and

optionally a port number), the Uri instance is created using only `relativeUri`.

The `dontEscape` parameter controls whether reserved characters are translated into escape sequences. This parameter should be set to `true` only if you are certain that all reserved characters in the URI have been escaped. Setting the value to `true` for a URI that has not been completely escaped can cause unexpected behavior. It is strongly recommended that you always set this parameter to `false`. If `dontEscape` is set to `false`, the constructor escapes any reserved characters by checking that all occurrences of percent (%) are followed by a valid escape sequence. If the character sequence following a percent is not valid, the percent is replaced by %25.

This constructor does not ensure that the Uri refers to an accessible resource.

Notes to Callers

Because of security concerns, your application should not call this constructor with URI strings from untrusted sources and with `dontEscape` set to `true`. Alternately, you can check a URI string for validity by calling the IsWellFormedOriginalString() method prior to calling this constructor.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Uri.SchemeDelimiter Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the characters that separate the communication protocol scheme from the address portion of the URI. This field is read-only.

C#

```
public static readonly string SchemeDelimiter;
```

Field Value

String

Examples

The following example creates a string from UriSchemeHttp, SchemeDelimiter, and an address. A Uri instance is then created from the string.

C#

```
            string address = "www.contoso.com";
            string uriString = String.Format("{0}{1}{2}/",
Uri.UriSchemeHttp, Uri.SchemeDelimiter, address);
 #if OLDMETHOD
            Uri result;
            if (Uri.TryParse(uriString, false, false, out result) == true)
                Console.WriteLine("{0} is a valid Uri", result.ToString());
            else
                Console.WriteLine("Uri not created");
#endif
            Uri result = new Uri(uriString);
            if (result.IsWellFormedOriginalString())
                Console.WriteLine("{0} is a well formed Uri", uriString);
            else
                Console.WriteLine("{0} is not a well formed Uri",
uriString);
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeFile Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is a pointer to a file. This field is read-only.

C#

```
public static readonly string UriSchemeFile;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeFile.

C#

```
Uri address2 =  new Uri("file://server/filename.ext");
if (address2.Scheme == Uri.UriSchemeFile)
    Console.WriteLine("Uri is a file");
```

Remarks

In .NET Framework version 1.1, a "`file:///path`" URI was translated to "`file:/path`". This has been corrected for version 2.0.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeFtp Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the File Transfer Protocol (FTP). This field is read-only.

C#

```
public static readonly string UriSchemeFtp;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeFtp.

C#

```
Uri address7 = new Uri("ftp://contoso/files/testfile.txt");
if (address7.Scheme == Uri.UriSchemeFtp)
    Console.WriteLine("Uri is Ftp protocol");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeFtps Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the File Transfer Protocol Secure (FTPS). This field is read-only.

C#

```
public static readonly string UriSchemeFtps;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeGopher Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the Gopher protocol. This field is read-only.

C#

```
public static readonly string UriSchemeGopher;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeGopher.

C#

```
Uri address6 = new Uri("gopher://example.contoso.com/");
if (address6.Scheme == Uri.UriSchemeGopher)
    Console.WriteLine("Uri is Gopher protocol");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeHttp Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the Hypertext Transfer Protocol (HTTP). This field is read-only.

C#

```
public static readonly string UriSchemeHttp;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeHttp.

C#

```
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine("address 1 {0} a valid scheme name",
      Uri.CheckSchemeName(address1.Scheme) ? " has" : " does not have");

if (address1.Scheme == Uri.UriSchemeHttp)
    Console.WriteLine("Uri is HTTP type");

Console.WriteLine(address1.HostNameType);
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeHttps Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the Secure Hypertext Transfer Protocol (HTTPS). This field is read-only.

C#

```
public static readonly string UriSchemeHttps;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeHttps.

C#

```
Uri address8 = new Uri("https://example.contoso.com");
if (address8.Scheme == Uri.UriSchemeHttps)
    Console.WriteLine("Uri is Https protocol.");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeMailto Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is an email address and is accessed through the Simple Mail Transport Protocol (SMTP). This field is read-only.

C#

```
public static readonly string UriSchemeMailto;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeMailto.

C#

```
Uri address3 = new Uri("mailto:user@contoso.com?subject=uri");
if (address3.Scheme == Uri.UriSchemeMailto)
    Console.WriteLine("Uri is an email address");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeNetPipe Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the NetPipe scheme used by Windows Communication Foundation (WCF). This field is read-only.

C#

```
public static readonly string UriSchemeNetPipe;
```

Field Value

String

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeNetTcp Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the NetTcp scheme used by Windows Communication Foundation (WCF). This field is read-only.

C#

```
public static readonly string UriSchemeNetTcp;
```

Field Value

String

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeNews Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.

C#

```
public static readonly string UriSchemeNews;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeNews.

C#

```
Uri address4 = new Uri("news:123456@contoso.com");
if (address4.Scheme == Uri.UriSchemeNews)
    Console.WriteLine("Uri is an Internet news group");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeNntp Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.

C#

```
public static readonly string UriSchemeNntp;
```

Field Value

String

Examples

The following example creates a Uri instance and determines whether the scheme is UriSchemeNntp.

C#

```
Uri address5 = new Uri("nntp://news.contoso.com/123456@contoso.com");
if (address5.Scheme == Uri.UriSchemeNntp)
    Console.WriteLine("Uri is nntp protocol");
```

Remarks

The NNTP Uri parsing errors in .NET Framework version 1.1 have been corrected.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.UriSchemeSftp Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the SSH File Transfer Protocol (SFTP). This field is read-only.

C#

```
public static readonly string UriSchemeSftp;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeSsh Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the Secure Socket Shell protocol (SSH). This field is read-only.

C#

```
public static readonly string UriSchemeSsh;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeTelnet Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the Telnet protocol. This field is read-only.

C#

```
public static readonly string UriSchemeTelnet;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeWs Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the WebSocket protocol (WS). This field is read-only.

C#

```
public static readonly string UriSchemeWs;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UriSchemeWss Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies that the URI is accessed through the WebSocket Secure protocol (WSS). This field is read-only.

C#

```
public static readonly string UriSchemeWss;
```

Field Value

String

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.AbsolutePath Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the absolute path of the URI.

```
C#

public string AbsolutePath { get; }
```

Property Value

String

The absolute path to the resource.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the path `/catalog/shownew.htm` to the console.

```
C#

Uri baseUri = new Uri("http://www.contoso.com/");
Uri myUri = new Uri(baseUri, "catalog/shownew.htm?date=today");

Console.WriteLine(myUri.AbsolutePath);
```

Remarks

The AbsolutePath property contains the path information that the server uses to resolve requests for information. Typically this is the path to the desired information on the

server's file system, although it also can indicate the application or script the server must run to provide the information.

The path information does not include the scheme, host name, or query portion of the URI.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.AbsoluteUri Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the absolute URI.

C#

```
public string AbsoluteUri { get; }
```

Property Value

String

The entire URI.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the complete contents of the Uri instance to the console. In the example shown, `http://www.contoso.com/catalog/shownew.htm?date=today` is written to the console.

C#

```
Uri baseUri= new Uri("http://www.contoso.com");
Uri myUri = new Uri(baseUri,"catalog/shownew.htm?date=today");
Console.WriteLine(myUri.AbsoluteUri);
```

Remarks

The AbsoluteUri property includes the entire URI stored in the Uri instance, including all fragments and query strings.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Authority Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Domain Name System (DNS) host name or IP address and the port number for a server.

```
C#

public string Authority { get; }
```

Property Value

String

The authority component of the URI represented by this instance.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the host name (`www.contoso.com`) and port number (8080) of the server to the console.

```
C#

Uri baseUri = new Uri("http://www.contoso.com:8080/");
Uri myUri = new Uri(baseUri,"shownew.htm?date=today");

Console.WriteLine(myUri.Authority);
```

Remarks

The Authority property is typically a server DNS host name or IP address. This property might include the service port number if it differs from the default port for the URI. If the Authority component contains reserved characters, these are escaped in the string value returned by this property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.DnsSafeHost Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a host name that, after being unescaped if necessary, is safe to use for DNS resolution.

```
C#
public string DnsSafeHost { get; }
```

Property Value

String

The host part of the URI in a format suitable for DNS resolution; or the original host string, if it is already suitable for resolution.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance from a string. It illustrates the difference between the value returned from Host, which returns the host name or address specified in the URI, and the value returned from DnsSafeHost, which returns an address that is safe to use in DNS resolution.

```
C#
// Create new Uri using a string address.
Uri address = new
Uri("http://[fe80::200:39ff:fe36:1a2d%254]/temp/example.htm");

// Make the address DNS safe.
```

```
// The following outputs "[fe80::200:39ff:fe36:1a2d]".
Console.WriteLine(address.Host);

// The following outputs "fe80::200:39ff:fe36:1a2d%254".
Console.WriteLine(address.DnsSafeHost);
```

As explained in Remarks, unescape the host name before resolving it. You can use the UnescapeDataString method to unescape the host name, and you can resolve it by calling the GetHostEntry method.

Remarks

For IPv6 addresses, the brackets ([]) are removed and the ScopeId property is set, if one was specified when this instance was constructed.

If you used an escaped string to construct this instance (for example, `"http://[fe80::200:39ff:fe36:1a2d%254]/temp/example.htm"`), then DnsSafeHost returns an escaped string. Unescape any escaped string returned from `DnsSafeHost` before using that string for DNS resolution (see the Example). If you used an invalid unescaped string to construct this instance (for example, "http://[fe80::200:39ff:fe36:1a2d%4]/temp/example.htm"), then DnsSafeHost returns an unescaped string.

The DnsSafeHost property is dependent on configuration settings in .NET Framework apps, as discussed later in this topic. The IdnHost property is provided as the preferred alternative to using DnsSafeHost, because IdnHost is guaranteed to always be DNS safe.

The DnsSafeHost property was extended in .NET Framework v3.5, 3.0 SP1, and 2.0 SP1 to provide International Resource Identifier (IRI) support based on RFC 3987. However, to ensure application compatibility with prior versions, you must specifically enable it in .NET Framework apps. To enable support for IRI, the following two changes are required:

1. Add the following line to the *machine.config* file under the .NET Framework 2.0 directory:

   `\<section name="uri" type="System.Configuration.UriSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />`

2. Specify whether you want Internationalized Domain Name (IDN) parsing applied to the domain name and whether IRI parsing rules should be applied. This can be done in the *machine.config* or in the *app.config* file. For example, add the following:

   ```
   XML
   ```

```xml
<configuration>
  <uri>
    <idn enabled="All" />
    <iriParsing enabled="true" />
  </uri>
</configuration>
```

Enabling IDN converts all Unicode labels in a domain name to their Punycode equivalents. Punycode names contain only ASCII characters and always start with the xn-- prefix. The reason for this is to support existing DNS servers on the Internet, since most DNS servers only support ASCII characters (see RFC 3940).

Enabling IDN only affects the value of the DnsSafeHost property.

There are three possible values for IDN depending on the DNS servers that are used:

- idn enabled = All

  This value will convert any Unicode domain names to their Punycode equivalents (IDN names).

- idn enabled = AllExceptIntranet

  This value will convert all external Unicode domain names to use the Punycode equivalents (IDN names). In this case to handle international names on the local Intranet, the DNS servers that are used for the Intranet should support Unicode names.

- idn enabled = None

  This value will not convert any Unicode domain names to use Punycode. This is the default value, which is consistent with the .NET Framework 2.0 behavior.

Enabling IRI parsing (iriParsing enabled = `true`) normalizes and checks characters according to the IRI rules in RFC 3987. The default value is `false` and normalizes and checks characters according to RFC 2396 and RFC 2732 (for IPv6 literals).

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Fragment Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the escaped URI fragment, including the leading '#' character if not empty.

C#

```
public string Fragment { get; }
```

Property Value

String

URI fragment information.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and writes the fragment information to the console.

C#

```
// Create Uri
Uri uriAddress = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine(uriAddress.Fragment);
Console.WriteLine("Uri {0} the default port ", uriAddress.IsDefaultPort ?
"uses" : "does not use");

Console.WriteLine("The path of this Uri is {0}",
uriAddress.GetLeftPart(UriPartial.Path));
Console.WriteLine("Hash code {0}", uriAddress.GetHashCode());
// The example displays output similar to the following:
//        #search
```

```
//        Uri uses the default port
//        The path of this Uri is http://www.contoso.com/index.htm
//        Hash code -988419291
```

Remarks

The Fragment property gets any text following a fragment marker (#) in the URI, including the fragment marker itself. Given the URI `http://www.contoso.com/index.htm#main`, the Fragment property would return #main.

The Fragment property is not considered in any Equals comparison.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Host Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the host component of this instance.

```
C#

public string Host { get; }
```

Property Value

String

The host name. This is usually the DNS host name or IP address of the server.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the host name (`www.contoso.com`) of the server to the console.

```
C#

Uri baseUri = new Uri("http://www.contoso.com:8080/");
Uri myUri = new Uri(baseUri, "shownew.htm?date=today");

Console.WriteLine(myUri.Host);
```

Remarks

Unlike the Authority property, this property value does not include the port number.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.HostNameType Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the type of the host name specified in the URI.

C#

```
public UriHostNameType HostNameType { get; }
```

Property Value

UriHostNameType

A member of the UriHostNameType enumeration.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and writes the HostNameType to the console.

C#

```
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine("address 1 {0} a valid scheme name",
      Uri.CheckSchemeName(address1.Scheme) ? " has" : " does not have");

if (address1.Scheme == Uri.UriSchemeHttp)
    Console.WriteLine("Uri is HTTP type");

Console.WriteLine(address1.HostNameType);
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IdnHost Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the RFC 3490 compliant International Domain Name of the host, using Punycode as appropriate. This string, after being unescaped if necessary, is safe to use for DNS resolution.

```
C#

public string IdnHost { get; }
```

Property Value

String

The hostname, formatted with Punycode according to the IDN standard.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Remarks

This property is provided for the use of lower-level networking protocols that require the domain name in Punycode form. If your code does not require that specific format, use Host for the hostname.

The deprecated DnsSafeHost property is dependent on *app.config* settings, which cannot be changed by Windows Store applications. IdnHost is provided as the preferred alternative to using DnsSafeHost, because IdnHost is guaranteed to always be DNS safe, no matter what the current *app.config* settings might be.

If you used an escaped string to construct this instance (for example, `"http://[fe80::200:39ff:fe36:1a2d%254]/temp/example.htm"`), then IdnHost returns an

escaped string. You should unescape any escaped string returned from IdnHost before using that string for DNS resolution. Be aware that if you used an invalid unescaped string to construct this instance (for example, "http://[fe80::200:39ff:fe36:1a2d%4]/temp/example.htm"), then IdnHost returns an unescaped string.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsAbsoluteUri Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the Uri instance is absolute.

```
C#
public bool IsAbsoluteUri { get; }
```

Property Value

Boolean

`true` if the Uri instance is absolute; otherwise, `false`.

Remarks

This property is `true` if the string or Uri instance that was passed into the constructor can be parsed as an absolute Uri instance, which contains a scheme, an authority, and a path. Otherwise, the Uri instance is treated as relative and might omit the scheme or other URI components.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.IsDefaultPort Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the port value of the URI is the default for this scheme.

C#

```
public bool IsDefaultPort { get; }
```

Property Value

Boolean

`true` if the value in the Port property is the default port for this scheme; otherwise, `false`.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and checks whether it uses the default port.

C#

```
// Create Uri
Uri uriAddress = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine(uriAddress.Fragment);
Console.WriteLine("Uri {0} the default port ", uriAddress.IsDefaultPort ?
"uses" : "does not use");

Console.WriteLine("The path of this Uri is {0}",
uriAddress.GetLeftPart(UriPartial.Path));
```

```
Console.WriteLine("Hash code {0}", uriAddress.GetHashCode());
// The example displays output similar to the following:
//        #search
//        Uri uses the default port
//        The path of this Uri is http://www.contoso.com/index.htm
//        Hash code -988419291
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsFile Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the specified Uri is a file URI.

C#

```
public bool IsFile { get; }
```

Property Value

Boolean

`true` if the Uri is a file URI; otherwise, `false`.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and determines whether it is a file URI.

C#

```
Uri uriAddress2 =  new Uri("file://server/filename.ext");
Console.WriteLine(uriAddress2.LocalPath);
Console.WriteLine("Uri {0} a UNC path", uriAddress2.IsUnc ? "is" : "is
not");
Console.WriteLine("Uri {0} a local host", uriAddress2.IsLoopback ? "is" :
"is not");
Console.WriteLine("Uri {0} a file", uriAddress2.IsFile ? "is" : "is not");
// The example displays the following output:
//    \\server\filename.ext
//    Uri is a UNC path
```

```
//    Uri is not a local host
//    Uri is a file
```

Remarks

The IsFile property is `true` when the Scheme property equals UriSchemeFile.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsLoopback Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the specified Uri references the local host.

C#

```
public bool IsLoopback { get; }
```

Property Value

Boolean

`true` if this Uri references the local host; otherwise, `false`.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and determines whether it references a local host.

C#

```
Uri uriAddress2 =  new Uri("file://server/filename.ext");
Console.WriteLine(uriAddress2.LocalPath);
Console.WriteLine("Uri {0} a UNC path", uriAddress2.IsUnc ? "is" : "is
not");
Console.WriteLine("Uri {0} a local host", uriAddress2.IsLoopback ? "is" :
"is not");
Console.WriteLine("Uri {0} a file", uriAddress2.IsFile ? "is" : "is not");
// The example displays the following output:
//    \\server\filename.ext
//    Uri is a UNC path
```

```
//    Uri is not a local host
//    Uri is a file
```

Remarks

IsLoopback returns `true` if the URI specified when this instance was created was 127.0.0.1, loopback, or localhost, or if the URI did not specify host information (for example, file:///c:Dir/file.txt). All other URIs return `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsUnc Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the specified Uri is a universal naming convention (UNC) path.

C#

```
public bool IsUnc { get; }
```

Property Value

Boolean

`true` if the Uri is a UNC path; otherwise, `false`.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and determines whether it is a UNC path.

C#

```
Uri uriAddress2 =  new Uri("file://server/filename.ext");
Console.WriteLine(uriAddress2.LocalPath);
Console.WriteLine("Uri {0} a UNC path", uriAddress2.IsUnc ? "is" : "is
not");
Console.WriteLine("Uri {0} a local host", uriAddress2.IsLoopback ? "is" :
"is not");
Console.WriteLine("Uri {0} a file", uriAddress2.IsFile ? "is" : "is not");
// The example displays the following output:
//    \\server\filename.ext
//    Uri is a UNC path
```

```
//    Uri is not a local host
//    Uri is a file
```

Remarks

The IsUnc property is `true` if the specified Uri instance is a UNC path (such as `\\server\folder`, //server, or `file://server/folder`). This property always returns `true` if the URI has the file:// scheme and specifies a host component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.LocalPath Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a local operating-system representation of a file name.

C#

```
public string LocalPath { get; }
```

Property Value

String

The local operating-system representation of a file name.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and writes the local path to the console.

C#

```
Uri uriAddress2 =  new Uri("file://server/filename.ext");
Console.WriteLine(uriAddress2.LocalPath);
Console.WriteLine("Uri {0} a UNC path", uriAddress2.IsUnc ? "is" : "is
not");
Console.WriteLine("Uri {0} a local host", uriAddress2.IsLoopback ? "is" :
"is not");
Console.WriteLine("Uri {0} a file", uriAddress2.IsFile ? "is" : "is not");
// The example displays the following output:
//    \\server\filename.ext
//    Uri is a UNC path
```

```
//    Uri is not a local host
//    Uri is a file
```

Remarks

The value returned by this property is unescaped. If the path is recognized as a Windows file path, all forward slashes (/) are replaced by backward slashes (\).

For the URI `file://computer/file.ext`, the absolute path is `/file.ext` and the local path is `\\computer\file.ext`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.OriginalString Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the original URI string that was passed to the Uri constructor.

C#

```
public string OriginalString { get; }
```

Property Value

String

The exact URI specified when this instance was constructed; otherwise, Empty.

Examples

The following example creates a new Uri instance from a string. It illustrates the difference between the value returned from OriginalString, which returns the string that was passed to the constructor, and from a call to ToString, which returns the canonical form of the string.

C#

```
// Create a new Uri from a string address.
Uri uriAddress = new
Uri("HTTP://www.ConToso.com:80//thick%20and%20thin.htm");

// Write the new Uri to the console and note the difference in the two
values.
// ToString() gives the canonical version. OriginalString gives the original
// string that was passed to the constructor.

// The following outputs "http://www.contoso.com//thick and thin.htm".
Console.WriteLine(uriAddress.ToString());

// The following outputs
```

```
"HTTP://www.ConToso.com:80//thick%20and%20thin.htm".
Console.WriteLine(uriAddress.OriginalString);
```

Remarks

If the URI specified to the constructor contained leading or trailing spaces, these spaces are preserved.

The value returned by this property differs from ToString and AbsoluteUri. ToString returns the canonically unescaped form of the URI. AbsoluteUri returns the canonically escaped form of the URI.

When International Resource Identifier (IRI) and Internationalized Domain Name (IDN) support are enabled, OriginalString returns the original non normalized string with Punycode host name if one was used to initialize the Uri instance. Punycode names contain only ASCII characters and always start with the xn-- prefix.

For more information on IRI support, see the Remarks section for the Uri class.

When a Uri object is serialized, the OriginalString is not preserved. The serialization process uses the fully escaped and canonicalized AbsoluteUri property when serializing. For a Uri that contains an IPv6 address, the IPv6 address and the scope ID are included in the serialized Uri object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

issues and pull requests. For more information, see our contributor guide.

Provide product feedback

Uri.PathAndQuery Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the AbsolutePath and Query properties separated by a question mark (?).

```
C#

public string PathAndQuery { get; }
```

Property Value

String

The AbsolutePath and Query properties separated by a question mark (?).

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the URI path (`/catalog/shownew.htm`) and query (`?date=today`) information to the console.

```
C#

Uri baseUri = new Uri("http://www.contoso.com/");
Uri myUri = new Uri(baseUri, "catalog/shownew.htm?date=today");

Console.WriteLine(myUri.PathAndQuery);
```

Remarks

The PathAndQuery property contains the absolute path on the server and the query information sent with the request. It is identical to concatenating the AbsolutePath and Query properties.

The PathAndQuery property is escaped according to RFC 2396 by default. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the PathAndQuery property is escaped according to RFC 3986 and RFC 3987.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Port Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the port number of this URI.

C#

```
public int Port { get; }
```

Property Value

Int32

The port number for this URI.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the URI port number to the console. In this case, the value is the default port number for HTTP, port 80.

C#

```
Uri baseUri = new Uri("http://www.contoso.com/");
Uri myUri = new Uri(baseUri,"catalog/shownew.htm?date=today");

Console.WriteLine(myUri.Port);
```

Remarks

The port number defines the protocol port used for contacting the server referenced in the URI. If a port is not specified as part of the URI, the Port property returns the default value for the protocol. If there is no default port number, this property returns -1.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Query Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets any query information included in the specified URI, including the leading '?' character if not empty.

C#

```
public string Query { get; }
```

Property Value

String

Any query information included in the specified URI.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the query `?date=today` to the console.

C#

```
Uri baseUri = new Uri ("http://www.contoso.com/");
Uri myUri = new Uri (baseUri, "catalog/shownew.htm?date=today");

Console.WriteLine (myUri.Query);
```

Remarks

The Query property contains any query information included in the URI. Query information is separated from the path information by a question mark (?) and continues to the end of the URI. The query information returned includes the leading question mark.

The query information is escaped according to RFC 2396 by default. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the query information is escaped according to RFC 3986 and RFC 3987.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Scheme Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the scheme name for this URI.

```
C#
public string Scheme { get; }
```

Property Value

String

The scheme for this URI, converted to lowercase.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example writes the scheme name (http) to the console for the `http://www.contoso.com/` URI.

```
C#
Uri baseUri = new Uri("http://www.contoso.com/");
Uri myUri = new Uri(baseUri, "catalog/shownew.htm?date=today");

Console.WriteLine(myUri.Scheme);
```

Remarks

The Scheme property returns the scheme used to initialize the Uri instance. This property does not indicate that the scheme used to initialize the Uri instance was recognized.

The following table shows examples of some possible values returned by the Scheme property.

Expand table

Scheme	Description
file	The resource is a file on the local computer.
ftp	The resource is accessed through FTP.
gopher	The resource is accessed through the Gopher protocol.
http	The resource is accessed through HTTP.
https	The resource is accessed through SSL-encrypted HTTP.
ldap	The resource is accessed through the LDAP protocol.
mailto	The resource is an email address and accessed through the SMTP protocol.
net.pipe	The resource is accessed through a named pipe.
net.tcp	The resource is accessed from TCP endpoint.
news	The resource is accessed through the NNTP protocol.
nntp	The resource is accessed through the NNTP protocol.
telnet	The resource is accessed through the TELNET protocol.
uuid	The resource is accessed through a unique UUID endpoint name for communicating with a service.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.Segments Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an array containing the path segments that make up the specified URI.

```
C#

public string[] Segments { get; }
```

Property Value

String[]

The path segments that make up the specified URI.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance with 3 segments and displays the segments on the screen.

```
C#

Uri uriAddress1 = new Uri("http://www.contoso.com/title/index.htm");
Console.WriteLine("The parts are {0}, {1}, {2}", uriAddress1.Segments[0],
uriAddress1.Segments[1], uriAddress1.Segments[2]);
```

Remarks

The Segments property returns an array of strings containing the "segments" (substrings) that form the URI's absolute path. The first segment is obtained by parsing

the absolute path from its first character until you reach a slash (/) or the end of the path. Each additional segment begins at the first character after the preceding segment, and terminates with the next slash or the end of the path. (A URI's absolute path contains everything after the host and port and before the query and fragment.)

The following example shows the absolute path and segments for two URIs. The second example illustrates that the fragment and query are not part of the absolute path and therefore are not segments.

Absolute URI: `http://www.contoso.com/Chapters/Chapter1/Sections/Section1.htm`

Absolute path: /Chapters/Chapter1/Sections/Section1.htm

Segments:

- /
- Chapters/
- Chapter1/
- Sections/
- Section1.htm

Absolute URI: `http://www.contoso.com/Chapters/Chapter1/Sections/Section1.htm#page1?answer=NO`

Absolute path: /Chapters/Chapter1/Sections/Section1.htm

Segments:

- /
- Chapters/
- Chapter1/
- Sections/
- Section1.htm

Note that because the absolute path starts with a '/', the first segment contains it and nothing else.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UserEscaped Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a value that indicates whether the URI string was completely escaped before the Uri instance was created.

```
C#

public bool UserEscaped { get; }
```

Property Value

Boolean

`true` if the `dontEscape` parameter was set to `true` when the Uri instance was created; otherwise, `false`.

Examples

The following example creates a Uri instance and determines whether it was fully escaped when it was created.

```
C#

Uri uriAddress = new Uri ("http://user:password@www.contoso.com/index.htm
");
Console.WriteLine(uriAddress.UserInfo);
Console.WriteLine("Fully Escaped {0}", uriAddress.UserEscaped ? "yes" :
"no");
```

Remarks

The UserEscaped property is set to `true` to indicate that the string used to create the Uri instance was completely escaped before it was passed to the constructor; that is, the `dontEscape` parameter of the constructor call was set to `true`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UserInfo Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the user name, password, or other user-specific information associated with the specified URI.

C#

```
public string UserInfo { get; }
```

Property Value

String

The user information associated with the URI. The returned value does not include the '@' character reserved for delimiting the user information part of the URI.

Exceptions

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates a Uri instance and writes the user information to the console.

C#

```
Uri uriAddress = new Uri ("http://user:password@www.contoso.com/index.htm
");
Console.WriteLine(uriAddress.UserInfo);
Console.WriteLine("Fully Escaped {0}", uriAddress.UserEscaped ? "yes" :
"no");
```

Remarks

The value returned by this property is usually in the format "userName:password".

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Canonicalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.Canonicalize has been deprecated and is not supported.

Converts the internally stored URI to canonical form.

C#

```
[System.Obsolete("Uri.Canonicalize has been deprecated and is not
supported.")]
protected virtual void Canonicalize ();
```

Attributes ObsoleteAttribute

Exceptions

InvalidOperationException

This instance represents a relative URI, and this method is valid only for absolute URIs.

UriFormatException

The URI is incorrectly formed.

Remarks

Stores the canonical version of the URI internally.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.CheckHostName(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified host name is a valid DNS name.

```
C#

public static UriHostNameType CheckHostName (string? name);
```

Parameters

name String

The host name to validate. This can be an IPv4 or IPv6 address or an Internet host name.

Returns

UriHostNameType

The type of the host name. If the type of the host name cannot be determined or if the host name is `null` or a zero-length string, this method returns Unknown.

Examples

The following example checks whether the host name is valid.

```
C#

Console.WriteLine(Uri.CheckHostName("www.contoso.com"));
```

Remarks

The CheckHostName method checks that the host name provided meets the requirements for a valid Internet host name. It does not, however, perform a host-name lookup to verify the existence of the host.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.CheckSchemeName(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the specified scheme name is valid.

C#

```
public static bool CheckSchemeName (string? schemeName);
```

Parameters

schemeName String

The scheme name to validate.

Returns

Boolean

`true` if the scheme name is valid; otherwise, `false`.

Examples

The following example creates a Uri instance and checks whether the scheme name is valid.

C#

```
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine("address 1 {0} a valid scheme name",
      Uri.CheckSchemeName(address1.Scheme) ? " has" : " does not have");

if (address1.Scheme == Uri.UriSchemeHttp)
    Console.WriteLine("Uri is HTTP type");

Console.WriteLine(address1.HostNameType);
```

Remarks

This method checks the scheme name for validity according to RFC 2396 by default. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, this method checks the scheme name for validity according to RFC 3986. The scheme name must begin with a letter and must contain only letters, digits, and the characters ".", "+", or "-".

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.CheckSecurity Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> **Caution**
>
> Uri.CheckSecurity has been deprecated and is not supported.

Calling this method has no effect.

C#

```
[System.Obsolete("Uri.CheckSecurity has been deprecated and is not supported.")]
protected virtual void CheckSecurity ();
```

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Uri.Compare(Uri, Uri, UriComponents, UriFormat, StringComparison) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the specified parts of two URIs using the specified comparison rules.

C#

```
public static int Compare (Uri? uri1, Uri? uri2, UriComponents
partsToCompare, UriFormat compareFormat, StringComparison comparisonType);
```

Parameters

`uri1` Uri

The first URI.

`uri2` Uri

The second URI.

`partsToCompare` UriComponents

A bitwise combination of the UriComponents values that specifies the parts of `uri1` and `uri2` to compare.

`compareFormat` UriFormat

One of the enumeration values that specifies the character escaping used when the URI components are compared.

`comparisonType` StringComparison

One of the enumeration values that specifies the culture, case, and sort rules for the comparison.

Returns

Int32

A value that indicates the lexical relationship between the compared Uri components.

Expand table

Value	Meaning
Less than zero	`uri1` is less than `uri2`.
Zero	`uri1` equals `uri2`.
Greater than zero	`uri1` is greater than `uri2`.

Exceptions

ArgumentException

`comparisonType` is not a valid StringComparison value.

Remarks

If both `uri1` and `uri2` are `null`, this method returns 0. When comparing URI values, a relative URI is always less than an absolute URI, and a non-null URI is always greater than a null URI. For cases where both `uri1` and `uri2` are not `null` and are both relative URIs or both absolute URIs, the String.Compare(String, String, StringComparison) method performs the comparison.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Uri.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares two Uri instances for equality.

C#

```
public override bool Equals (object? comparand);
```

Parameters

comparand Object

The URI or a URI identifier to compare with the current instance.

Returns

Boolean

`true` if the two instances represent the same URI; otherwise, `false`.

Examples

This example creates two Uri instances from strings and compares them to determine whether they represent the same value. `address1` and `address2` are the same because the Fragment portion is ignored for this comparison. The outcome is written to the console.

C#

```
// Create some Uris.
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
Uri address2 = new Uri("http://www.contoso.com/index.htm");
if (address1.Equals(address2))
    Console.WriteLine("The two addresses are equal");
else
```

```
    Console.WriteLine("The two addresses are not equal");
// Will output "The two addresses are equal"
```

Remarks

The Equals method compares the two instances without regard to user information (UserInfo) and fragment (Fragment) parts that they might contain. For example, given the URIs `http://www.contoso.com/index.htm#search` and `http://user:password@www.contoso.com/index.htm`, the Equals method would return `true`.

If one Uri instance is formed with a Unicode host name and `comparand` parameter contains a Uri instance or identifier that is formed with a host name that has the equivalent Punycode host name, then Equals returns `true` only if International Resource Identifier (IRI) and Internationalized Domain Name (IDN) support are enabled. Punycode names contain only ASCII characters and always start with the xn-- prefix.

For more information on IRI support, see the Remarks section for the Uri class.

> **Note**
>
> In the .NET Framework versions 1.0 and 1.1, the **Query** is also ignored.

> **Note**
>
> The **Equals** method can be overridden in a derived class; use caution as a malicious entity could modify the method. You should not use this method to perform security checks unless you know that this instance came from a trusted source.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.Escape Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.Escape has been deprecated and is not supported.

Converts any unsafe or reserved characters in the path component to their hexadecimal character representations.

C#

```
[System.Obsolete("Uri.Escape has been deprecated and is not supported.")]
protected virtual void Escape ();
```

Attributes ObsoleteAttribute

Exceptions

UriFormatException

The URI passed from the constructor is invalid. This exception can occur if a URI has too many characters or the URI is relative.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.EscapeDataString(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a string to its escaped representation.

```
C#
public static string EscapeDataString (string stringToEscape);
```

Parameters

`stringToEscape` String

The string to escape.

Returns

String

The escaped representation of `stringToEscape`.

Exceptions

ArgumentNullException

`stringToEscape` is `null`.

UriFormatException

Note: In the .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

The length of `stringToEscape` exceeds 32766 characters.

Remarks

By default, the EscapeDataString method converts all characters except for RFC 2396 unreserved characters to their hexadecimal representation. If International Resource

Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the EscapeDataString method converts all characters, except for RFC 3986 unreserved characters, to their hexadecimal representation. All Unicode characters are converted to UTF-8 format before being escaped.

This method assumes that `stringToEscape` has no escape sequences in it.

By default, the string is escaped according to RFC 2396. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the string is escaped according to RFC 3986 and RFC 3987. See these RFCs for a definition of reserved and unreserved characters.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.EscapeString(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.EscapeString has been deprecated. Use GetComponents() or Uri.EscapeDataString to escape a Uri component or a string.

Converts a string to its escaped representation.

C#

```
[System.Obsolete("Uri.EscapeString has been deprecated. Use GetComponents()
or Uri.EscapeDataString to escape a Uri component or a string.")]
protected static string EscapeString (string? str);
```

Parameters

`str` String

The string to transform to its escaped representation.

Returns

String

The escaped representation of the string.

Attributes ObsoleteAttribute

Remarks

The EscapeString method converts RFC 2396 reserved characters and all characters with a character value greater than 127 to hexadecimal representation. All Unicode characters are converted to UTF-8 format before being escaped.

By default, the string is escaped according to RFC 2396. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the string is escaped according to RFC 3986 and RFC 3987.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.EscapeUriString(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.EscapeUriString can corrupt the Uri string in some cases. Consider using Uri.EscapeDataString for query string components instead.

Converts a URI string to its escaped representation.

C#

```
[System.Obsolete("Uri.EscapeUriString can corrupt the Uri string in some cases. Consider using Uri.EscapeDataString for query string components instead.", DiagnosticId="SYSLIB0013", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public static string EscapeUriString (string stringToEscape);
```

Parameters

`stringToEscape` String

The string to escape.

Returns

String

The escaped representation of `stringToEscape`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`stringToEscape` is `null`.

UriFormatException

The length of `stringToEscape` exceeds 32766 characters.

```
    Note: In <a href="https://docs.microsoft.com/previous-
versions/br230232(v=vs.110)">.NET for Windows Store apps</a> or the <a
href="https://docs.microsoft.com/dotnet/standard/cross-platform/cross-
platform-development-with-the-portable-class-library">Portable Class
Library</a>, catch the base class exception, <xref data-throw-if-not-
resolved="true" uid="System.FormatException"></xref>, instead.
```

Remarks

Use the EscapeUriString method to prepare an unescaped URI string to be a parameter to the Uri constructor.

By default, the EscapeUriString method converts all characters, except RFC 2396 unreserved characters, to their hexadecimal representation. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the EscapeUriString method converts all characters, except for RFC 3986 unreserved characters, to their hexadecimal representation. All Unicode characters are converted to UTF-8 format before being escaped.

This method assumes that `stringToEscape` has no escape sequences in it.

By default, the string is escaped according to RFC 2396. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the string is escaped according to RFC 3986 and RFC 3987. See these RFCs for a definition of reserved and unreserved characters.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5 *(6, 7, 8, 9)*
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.FromHex(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the decimal value of a hexadecimal digit.

C#

```
public static int FromHex (char digit);
```

Parameters

`digit` Char

The hexadecimal digit (0-9, a-f, A-F) to convert.

Returns

Int32

A number from 0 to 15 that corresponds to the specified hexadecimal digit.

Exceptions

ArgumentException

`digit` is not a valid hexadecimal digit (0-9, a-f, A-F).

Examples

The following example determines whether a character is a hexadecimal character and, if it is, writes the corresponding decimal value to the console.

C#

```
char  testChar = 'e';
if (Uri.IsHexDigit(testChar) == true)
    Console.WriteLine("'{0}' is the hexadecimal representation of {1}",
testChar, Uri.FromHex(testChar));
```

```
else
    Console.WriteLine("'{0}' is not a hexadecimal character", testChar);

string returnString = Uri.HexEscape(testChar);
Console.WriteLine("The hexadecimal value of '{0}' is {1}", testChar,
returnString);
```

Remarks

The FromHex method converts a character representing a hexadecimal digit (0-9, a-f, A-F) to its decimal value (0 to 15). If `digit` is not a valid hexadecimal digit, an ArgumentException exception is thrown.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.GetComponents(UriComponents, UriFormat) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the specified components of the current instance using the specified escaping for special characters.

C#

```
public string GetComponents (UriComponents components, UriFormat format);
```

Parameters

`components` UriComponents

A bitwise combination of the UriComponents values that specifies which parts of the current instance to return to the caller.

`format` UriFormat

One of the enumeration values that controls how special characters are escaped.

Returns

String

The components of the current instance.

Exceptions

ArgumentOutOfRangeException

`components` is not a combination of valid UriComponents values.

InvalidOperationException

The current Uri is not an absolute URI. Relative URIs cannot be used with this method.

Remarks

The Query, Fragment, Scheme, UserInfo, Host, Port, and Path components do not include the delimiter. You can combine the KeepDelimiter flag (using the bitwise OR operator) with any of those values to get the value with the delimiter. For all other UriComponents values, and combinations of values, the delimiters are included in the returned value.

The components are returned in the order that they appear in the URI. For example, if Scheme is specified, it appears first.

When International Resource Identifier (IRI) and Internationalized Domain Name (IDN) support are enabled, the number of characters returned in the String increases. Punycode names used to support IRI contain only ASCII characters and always start with the xn-- prefix. When IRI and IDN are enabled, Unicode surrogate characters are handled correctly by the GetComponents method.

For more information on IRI support, see the Remarks section for the Uri class.

> **Note**
>
> If the **GetComponents** method is called with `format` set to **Unescaped** , you cannot use the return value as an argument to a **Uri** constructor to create an equivalent **Uri**.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Uri.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the hash code for the URI.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash value generated for this URI.

Examples

The following example creates a Uri instance and writes the hash code to the console.

C#

```
// Create Uri
Uri uriAddress = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine(uriAddress.Fragment);
Console.WriteLine("Uri {0} the default port ", uriAddress.IsDefaultPort ?
"uses" : "does not use");

Console.WriteLine("The path of this Uri is {0}",
uriAddress.GetLeftPart(UriPartial.Path));
Console.WriteLine("Hash code {0}", uriAddress.GetHashCode());
// The example displays output similar to the following:
//        #search
//        Uri uses the default port
//        The path of this Uri is http://www.contoso.com/index.htm
//        Hash code -988419291
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.GetLeftPart(UriPartial) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the specified portion of a Uri instance.

C#

```
public string GetLeftPart (UriPartial part);
```

Parameters

`part` UriPartial

One of the enumeration values that specifies the end of the URI portion to return.

Returns

String

The specified portion of the Uri instance.

Exceptions

InvalidOperationException

The current Uri instance is not an absolute instance.

ArgumentException

The specified `part` is not valid.

Examples

The following example creates a Uri instance and writes the path to the console.

C#

```
// Create Uri
Uri uriAddress = new Uri("http://www.contoso.com/index.htm#search");
Console.WriteLine(uriAddress.Fragment);
Console.WriteLine("Uri {0} the default port ", uriAddress.IsDefaultPort ? "uses" : "does not use");

Console.WriteLine("The path of this Uri is {0}", uriAddress.GetLeftPart(UriPartial.Path));
Console.WriteLine("Hash code {0}", uriAddress.GetHashCode());
// The example displays output similar to the following:
//        #search
//        Uri uses the default port
//        The path of this Uri is http://www.contoso.com/index.htm
//        Hash code -988419291
```

Remarks

The GetLeftPart method returns a string containing the leftmost portion of the URI string, ending with the portion specified by `part`.

GetLeftPart includes delimiters in the following cases:

- Scheme includes the scheme delimiter.
- Authority does not include the path delimiter.
- Path includes the path delimiter and any delimiters in the original URI up to the query or fragment delimiter.
- Query includes the Path, plus the query and its delimiter.

The following examples show a URI and the results of calling GetLeftPart with Scheme, Authority, Path, or Query.

Expand table

URI	Scheme	Authority	Path	Query
`http://www.contoso.com/index.htm?date=today`	`http://`	`http://www.contoso.com`	`http://www.contoso.com/index.htm`	`http://www.contoso.com/index.h date=today`
`http://www.contoso.com/index.htm#main`	`http://`	`http://www.contoso.com`	`http://www.contoso.com/index.htm`	`http://www.contoso.com/index.h`
`mailto:user@contoso.com?subject=uri`	`mailto:`	`<none>`	`mailto:user@contoso.com`	`mailto:user@contoso.com?subje`
`nntp://news.contoso.com/123456@contoso.com`	`nntp://`	`nntp://news.contoso.com`	`nntp://news.contoso.com/123456@contoso.com`	`nntp://news.contoso.com/12345`
`news:123456@contoso.com`	`news:`	`<none>`	`news:123456@contoso.com`	`news:123456@contoso.com`
`file://server/filename.ext`	`file://`	`file://server`	`file://server/filename.ext`	`file://server/filename.ext`

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the data needed to serialize the current instance.

C#

```
protected void GetObjectData (System.Runtime.Serialization.SerializationInfo
serializationInfo, System.Runtime.Serialization.StreamingContext
streamingContext);
```

Parameters

`serializationInfo` SerializationInfo

The information required to serialize the Uri.

`streamingContext` StreamingContext

An object that contains the source and destination of the serialized stream associated with the Uri.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.HexEscape(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified character into its hexadecimal equivalent.

C#

```
public static string HexEscape (char character);
```

Parameters

`character` Char

The character to convert to hexadecimal representation.

Returns

String

The hexadecimal representation of the specified character.

Exceptions

ArgumentOutOfRangeException

`character` is greater than 255.

Examples

The following example converts a character to its hexadecimal equivalent and writes it to the console.

C#

```
char  testChar = 'e';
if (Uri.IsHexDigit(testChar) == true)
    Console.WriteLine("'{0}' is the hexadecimal representation of {1}",
testChar, Uri.FromHex(testChar));
```

```
else
    Console.WriteLine("'{0}' is not a hexadecimal character", testChar);

string returnString = Uri.HexEscape(testChar);
Console.WriteLine("The hexadecimal value of '{0}' is {1}", testChar,
returnString);
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.HexUnescape(String, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a specified hexadecimal representation of a character to the character.

```
C#
public static char HexUnescape (string pattern, ref int index);
```

Parameters

pattern String

The hexadecimal representation of a character.

index Int32

The location in `pattern` where the hexadecimal representation of a character begins.

Returns

Char

The character represented by the hexadecimal encoding at position `index`. If the character at `index` is not hexadecimal encoded, the character at `index` is returned. The value of `index` is incremented to point to the character following the one returned.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0 or greater than or equal to the number of characters in `pattern`.

Remarks

The following code example determines whether a character is hexadecimal encoded and, if so, writes the equivalent character to the console.

C#

```
string testString = "%75";
int index = 0;
if (Uri.IsHexEncoding(testString, index))
     Console.WriteLine("The character is {0}", Uri.HexUnescape(testString,
ref index));
else
     Console.WriteLine("The character is not hexadecimal encoded");
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsBadFileSystemCharacter(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.IsBadFileSystemCharacter has been deprecated and is not supported.

Indicates whether a character is invalid in a file system name.

C#

```
[System.Obsolete("Uri.IsBadFileSystemCharacter has been deprecated and is not supported.")]
protected virtual bool IsBadFileSystemCharacter (char character);
```

Parameters

`character` Char

The Char to test.

Returns

Boolean

`true` if the specified character is invalid; otherwise, `false`.

Attributes ObsoleteAttribute

Remarks

The character is analyzed according to the rules for the NTFS file system.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsBaseOf(Uri) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the current Uri instance is a base of the specified Uri instance.

```
C#

public bool IsBaseOf (Uri uri);
```

Parameters

uri Uri

The specified URI to test.

Returns

Boolean

`true` if the current Uri instance is a base of `uri`; otherwise, `false`.

Exceptions

ArgumentNullException

`uri` is `null`.

Examples

This example creates a Uri instance that represents a base Uri instance. It then creates a second Uri instance from a string. It calls IsBaseOf to determine whether the base instance is the base of the second instance. The outcome is written to the console.

```
C#

// Create a base Uri.
Uri baseUri = new Uri("http://www.contoso.com/");
```

```
// Create a new Uri from a string.
Uri uriAddress = new Uri("http://www.contoso.com/index.htm?date=today");

// Determine whether BaseUri is a base of UriAddress.
if (baseUri.IsBaseOf(uriAddress))
    Console.WriteLine("{0} is the base of {1}", baseUri, uriAddress);
```

Remarks

IsBaseOf is used to compare the current Uri instance to a specified Uri to determine whether this URI is a base for the specified Uri. When comparing two Uri objects to determine a base relationship, the user information (UserInfo) is not evaluated. When comparing two URIs (uri1 and uri2), uri1 is the base of uri2 if, when you ignore everything in uri1 and uri2 after the last slash (/), uri2 starts with exactly uri1. Using http://host/path/path/file?query as the base URI, the following table shows whether it is a base for other URIs.

Expand table

URI	http://host/path/path/file?query is base of
http://host/path/path/file/	yes
http://host/path/path/#fragment	yes
http://host/path/path/MoreDir/"	yes
http://host/path/path/OtherFile?Query	yes
http://host/path/path/	yes
http://host/path/path/file	yes
http://host/path/path	no
http://host/path/path?query	no
http://host/path/path#Fragment	no
http://host/path/path2/	no
://host/path/path2/MoreDir	no
http://host/path/File	no

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsExcludedCharacter(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.IsExcludedCharacter has been deprecated and is not supported.

Determines whether the specified character should be escaped.

C#

```
[System.Obsolete("Uri.IsExcludedCharacter has been deprecated and is not supported.")]
protected static bool IsExcludedCharacter (char character);
```

Parameters

`character` Char

The character to test.

Returns

Boolean

`true` if the specified character should be escaped; otherwise, `false`.

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsHexDigit(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether a specified character is a valid hexadecimal digit.

C#

```
public static bool IsHexDigit (char character);
```

Parameters

`character` Char

The character to validate.

Returns

Boolean

`true` if the character is a valid hexadecimal digit; otherwise, `false`.

Examples

The following example determines whether a character is a hexadecimal character and, if it is, writes the corresponding decimal value to the console.

C#

```
char  testChar = 'e';
if (Uri.IsHexDigit(testChar) == true)
    Console.WriteLine("'{0}' is the hexadecimal representation of {1}",
testChar, Uri.FromHex(testChar));
else
    Console.WriteLine("'{0}' is not a hexadecimal character", testChar);

string returnString = Uri.HexEscape(testChar);
Console.WriteLine("The hexadecimal value of '{0}' is {1}", testChar,
returnString);
```

Remarks

Hexadecimal digits are the digits 0 to 9 and the letters A-F or a-f.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsHexEncoding(String, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether a character in a string is hexadecimal encoded.

```
C#
public static bool IsHexEncoding (string pattern, int index);
```

Parameters

pattern String

The string to check.

index Int32

The location in `pattern` to check for hexadecimal encoding.

Returns

Boolean

`true` if `pattern` is hexadecimal encoded at the specified location; otherwise, `false`.

Examples

The following code example determines whether a character is hexadecimal encoded and, if so, writes the equivalent character to the console.

```
C#
string testString = "%75";
int index = 0;
if (Uri.IsHexEncoding(testString, index))
     Console.WriteLine("The character is {0}", Uri.HexUnescape(testString,
ref index));
```

```
else
    Console.WriteLine("The character is not hexadecimal encoded");
```

Remarks

The IsHexEncoding method checks for hexadecimal encoding that follows the pattern "%hexhex" in a string, where "hex" is a digit from 0 to 9 or a letter from A-F (case-insensitive).

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsReservedCharacter(Char) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.IsReservedCharacter has been deprecated and is not supported.

Determines whether the specified character is a reserved character.

C#

```
[System.Obsolete("Uri.IsReservedCharacter has been deprecated and is not supported.")]
protected virtual bool IsReservedCharacter (char character);
```

Parameters

`character` Char

The character to test.

Returns

Boolean

`true` if the specified character is a reserved character otherwise, `false`.

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IsWellFormedOriginalString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the string used to construct this Uri was well-formed and does not require further escaping.

```
C#
public bool IsWellFormedOriginalString ();
```

Returns

Boolean

`true` if the string was well-formed; otherwise, `false`.

Remarks

The string is considered to be well-formed in accordance with RFC 2396 and RFC 2732 by default. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing is enabled, the string is considered to be well-formed in accordance with RFC 3986 and RFC 3987

The string is considered poorly formed, causing the method to return false, if any of the following conditions occur.

Expand table

Error	Example
The string is not correctly escaped.	`http://www.contoso.com/path???/file name`
The string is an absolute Uri that represents an implicit file Uri.	`c:\\directory\filename`
The string is an absolute URI that is missing a slash before the path.	`file://c:/directory/filename`

Error	Example
The string contains unescaped backslashes even if they are treated as forward slashes.	`http:\\host/path/file`
The string represents a hierarchical absolute Uri and does not contain "://".	`www.contoso.com/path/file`
The parser for the Uri.Scheme indicates that the original string was not well-formed.	The example depends on the scheme of the URI.

By default, the string used to construct this Uri are considered well-formed in accordance with RFC 2396 and RFC 2732.

When International Resource Identifier (IRI) and Internationalized Domain Name (IDN) support are enabled, the string used to construct this Uri are considered well-formed in accordance with RFC 3986 and RFC 3987. Punycode names used to support IRI contain only ASCII characters and always start with the xn-- prefix.

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Internet Engineering Task Force (IETF)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Uri.IsWellFormedUriString(String, UriKind) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the string is well-formed by attempting to construct a URI with the string and ensures that the string does not require further escaping.

C#

```
public static bool IsWellFormedUriString (string? uriString, UriKind
uriKind);
```

Parameters

`uriString` String

The string used to attempt to construct a Uri.

`uriKind` UriKind

The type of the Uri in `uriString`.

Returns

Boolean

`true` if the string was well-formed; otherwise, `false`.

Remarks

In versions of .NET before version 4.5, by default the string is considered well-formed in accordance with RFC 2396 and RFC 2732. If International Resource Identifiers (IRIs) or Internationalized Domain Name (IDN) parsing are enabled, the string is considered well-formed in accordance with RFC 3986 and RFC 3987.

Beginning in .NET 4.5, strings are always considered well-formed in accordance with RFC 3986 and RFC 3987, whether or not IRI or IDN are enabled. Note, however, that this is only true for apps that target .NET 4.5 or later. Apps that target .NET 4.0 invoke compatibility code, and experience the old (pre-4.5) behavior.

The string is considered poorly formed, causing the method to return false, if any of the following conditions occur

Expand table

Error	Example
The string is not correctly escaped.	`http://www.contoso.com/path???/filename`
The string is an absolute Uri that represents an implicit file Uri.	`c:\\directory\filename`
The string is an absolute URI that is missing a slash before the path.	`file://c:/directory/filename`
The string contains unescaped backslashes even if they will be treated as forward slashes	`http:\\\host/path/file`
The string represents a hierarchical absolute Uri and does not contain "://"	`www.contoso.com/path/file`
The parser for the Scheme indicates that the original string was not well-formed.	The example depends on the scheme of the URI.
Beginning with .NET 4.5, relative URIs with a colon (':') in their first segment are not considered well-formed.	`2013.05.29_14:33:41`

For more information on IRI support, see the Remarks section for the Uri class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Internet Engineering Task Force (IETF)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.MakeRelative(Uri) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.MakeRelative has been deprecated. Use MakeRelativeUri(Uri uri) instead.

Determines the difference between two Uri instances.

C#

```
[System.Obsolete("Uri.MakeRelative has been deprecated. Use MakeRelativeUri(Uri uri)
instead.")]
public string MakeRelative (Uri toUri);
```

Parameters

`toUri` Uri

The URI to compare to the current URI.

Returns

String

If the hostname and scheme of this URI instance and `toUri` are the same, then this method returns a String that represents a relative URI that, when appended to the current URI instance, yields the `toUri` parameter.

If the hostname or scheme is different, then this method returns a String that represents the `toUri` parameter.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`toUri` is `null`.

InvalidOperationException

This instance represents a relative URI, and this method is valid only for absolute URIs.

Examples

The following example creates 2 Uri instances. The difference in the path information is written to the console.

C#

```
// Create a base Uri.
Uri address1 = new Uri("http://www.contoso.com/");

// Create a new Uri from a string.
Uri address2 = new Uri("http://www.contoso.com/index.htm?date=today");

// Determine the relative Uri.
Console.WriteLine("The difference is {0}", address1.MakeRelativeUri(address2));
```

Remarks

The following table shows the URI instance, `toUri`, and the results of calling MakeRelative.

Expand table

Current URI instance	`toUri`	Return value
`http://www.contoso.com/`	`http://www.contoso.com/test/test.htm`	`test/test.htm`
`http://www.contoso.com/test1/`	`http://www.contoso.com/`	`../`
`http://www.contoso.com:8000/`	`http://www.contoso.com/test/test.htm`	`http://www.contoso.com/test/test.htm`
`http://username@www.contoso.com/`	`http://www.contoso.com/test1/test1.txt`	`test1/test1.txt`

User information, if present in the URI, is ignored.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.MakeRelativeUri(Uri) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines the difference between two Uri instances.

```
C#

public Uri MakeRelativeUri (Uri uri);
```

Parameters

uri Uri

The URI to compare to the current URI.

Returns

Uri

If the hostname and scheme of this URI instance and `uri` are the same, then this method returns a relative Uri that, when appended to the current URI instance, yields `uri`.

If the hostname or scheme is different, then this method returns a Uri that represents the `uri` parameter.

Exceptions

ArgumentNullException

`uri` is `null`.

InvalidOperationException

This instance represents a relative URI, and this property is valid only for absolute URIs.

Examples

The following example creates 2 Uri instances. The difference in the path information is written to the console.

```
C#

// Create a base Uri.
Uri address1 = new Uri("http://www.contoso.com/");
```

```
// Create a new Uri from a string.
Uri address2 = new Uri("http://www.contoso.com/index.htm?date=today");

// Determine the relative Uri.
Console.WriteLine("The difference is {0}", address1.MakeRelativeUri(address2));
```

Remarks

The following table shows the URI instance, `toUri`, and the results of calling MakeRelativeUri.

Expand table

Current URI instance	`toUri`	Return value
`http://www.contoso.com/`	`http://www.contoso.com/test/test.htm`	`test/test.htm`
`http://www.contoso.com/test1/`	`http://www.contoso.com/`	`../`
`http://www.contoso.com:8000/`	`http://www.contoso.com/test/test.htm`	`http://www.contoso.com/test/test.htm`
`http://username@www.contoso.com/`	`http://www.contoso.com/test1/test1.txt`	`test1/test1.txt`

User information, if present in the URI, is ignored.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.Parse has been deprecated and is not supported.

Parses the URI of the current instance to ensure it contains all the parts required for a valid URI.

C#

```
[System.Obsolete("Uri.Parse has been deprecated and is not supported.")]
protected virtual void Parse ();
```

Attributes ObsoleteAttribute

Exceptions

UriFormatException

The Uri passed from the constructor is invalid.

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets a canonical string representation for the specified Uri instance.

C#

```
public override string ToString ();
```

Returns

String

The unescaped canonical representation of the Uri instance. All characters are unescaped except #, ?, and %.

Examples

The following example creates a new Uri instance from a string. It illustrates the difference between the value returned from OriginalString, which returns the string that was passed to the constructor, and from a call to ToString, which returns the canonical form of the string.

C#

```
// Create a new Uri from a string address.
Uri uriAddress = new
Uri("HTTP://www.Contoso.com:80/thick%20and%20thin.htm");

// Write the new Uri to the console and note the difference in the two
values.
// ToString() gives the canonical version.  OriginalString gives the orginal
// string that was passed to the constructor.

// The following outputs "http://www.contoso.com/thick and thin.htm".
Console.WriteLine(uriAddress.ToString());
```

```
// The following outputs "HTTP://www.Contoso.com:80/thick%20and%20thin.htm".
Console.WriteLine(uriAddress.OriginalString);
```

Remarks

The string returned by this method does not contain port information when the port is the default port for the scheme.

> **Note**
>
> The string returned by the **ToString** method may contain control characters, which can corrupt the state of a console application. You can use the **GetComponents** method with the **UriFormat.SafeUnescaped** format to remove control characters from the returned string.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.TryCreate Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new Uri. Does not throw an exception if the Uri cannot be created.

Overloads

Expand table

TryCreate(Uri, Uri, Uri)	Creates a new Uri using the specified base and relative Uri instances.
TryCreate(String, UriKind, Uri)	Creates a new Uri using the specified String instance and a UriKind.
TryCreate(String, UriCreationOptions, Uri)	Creates a new Uri using the specified String instance and UriCreationOptions.
TryCreate(Uri, String, Uri)	Creates a new Uri using the specified base and relative String instances.

TryCreate(Uri, Uri, Uri)

Creates a new Uri using the specified base and relative Uri instances.

C#

```
public static bool TryCreate (Uri? baseUri, Uri? relativeUri, out Uri?
result);
```

Parameters

baseUri Uri

The base URI.

`relativeUri` Uri

The relative URI to add to the base Uri.

`result` Uri

When this method returns, contains a Uri constructed from `baseUri` and `relativeUri`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the Uri was successfully created; otherwise, `false`.

Exceptions

ArgumentNullException

`baseUri` is `null`.

Remarks

If this method returns `true`, the new Uri is in `result`.

This method constructs the URI, puts it in canonical form, and validates it. If an unhandled exception occurs, this method catches it. If you want to create a Uri and get exceptions use one of the Uri constructors.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryCreate(String, UriKind, Uri)

Creates a new Uri using the specified String instance and a UriKind.

```
C#

public static bool TryCreate (string? uriString, UriKind uriKind, out
Uri? result);
```

Parameters

`uriString` String

The string representation of the Uri.

`uriKind` UriKind

The type of the Uri.

`result` Uri

When this method returns, contains the constructed Uri.

Returns

Boolean

`true` if the Uri was successfully created; otherwise, `false`.

Remarks

If this method returns `true`, the new Uri is in `result`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryCreate(String, UriCreationOptions, Uri)

Creates a new Uri using the specified String instance and UriCreationOptions.

```
C#

public static bool TryCreate (string? uriString, in UriCreationOptions
creationOptions, out Uri? result);
```

Parameters

`uriString` String

The string representation of the Uri.

`creationOptions` UriCreationOptions

Options that control how the Uri is created and behaves.

`result` Uri

When this method returns, contains the constructed Uri.

Returns

Boolean

`true` if the Uri was successfully created; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

TryCreate(Uri, String, Uri)

Creates a new Uri using the specified base and relative String instances.

```
C#

public static bool TryCreate (Uri? baseUri, string? relativeUri, out Uri?
```

```
result);
```

Parameters

`baseUri` Uri

The base URI.

`relativeUri` String

The string representation of the relative URI to add to the base Uri.

`result` Uri

When this method returns, contains a Uri constructed from `baseUri` and `relativeUri`. This parameter is passed uninitialized.

Returns

Boolean

`true` if the Uri was successfully created; otherwise, `false`.

Remarks

If this method returns `true`, the new Uri is in `result`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.TryFormat(Span<Char>, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Attempts to format a canonical string representation for the Uri instance into the specified span.

C#

```
public bool TryFormat (Span<char> destination, out int charsWritten);
```

Parameters

destination Span<Char>

The span into which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.Unescape(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> Uri.Unescape has been deprecated. Use GetComponents() or Uri.UnescapeDataString() to unescape a Uri component or a string.

Converts the specified string by replacing any escape sequences with their unescaped representation.

C#

```
[System.Obsolete("Uri.Unescape has been deprecated. Use GetComponents() or Uri.UnescapeDataString() to unescape a Uri component or a string.")]
protected virtual string Unescape (string path);
```

Parameters

`path` String

The string to convert.

Returns

String

The unescaped value of the `path` parameter.

Attributes ObsoleteAttribute

Applies to

Product	Versions *(Obsolete)*
.NET	*(Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	1.1 *(2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.UnescapeDataString(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts a string to its unescaped representation.

C#

```
public static string UnescapeDataString (string stringToUnescape);
```

Parameters

stringToUnescape String

The string to unescape.

Returns

String

The unescaped representation of `stringToUnescape`.

Exceptions

ArgumentNullException

`stringToUnescape` is `null`.

Examples

The following code example unescapes a URI, and then converts any plus characters ("+") into spaces.

C#

```
String DataString = Uri.UnescapeDataString(".NET+Framework");
Console.WriteLine("Unescaped string: {0}", DataString);
```

```
String PlusString = DataString.Replace('+',' ');
Console.WriteLine("plus to space string: {0}", PlusString);
```

Remarks

You should use this method with care. Unescaping a string that has been previously unescaped can lead to ambiguities and errors.

Many Web browsers escape spaces inside of URIs into plus ("+") characters; however, the UnescapeDataString method does not convert plus characters into spaces because this behavior is not standard across all URI schemes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- EscapeDataString(String)
- EscapeUriString(String)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Equality(Uri, Uri) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two Uri instances have the same value.

```
C#
public static bool operator == (Uri? uri1, Uri? uri2);
```

Parameters

uri1 Uri

A URI to compare with `uri2`.

uri2 Uri

A URI to compare with `uri1`.

Returns

Boolean

`true` if the Uri instances are equivalent; otherwise, `false`.

Examples

This example creates three Uri instances from strings and compares them to determine whether they represent the same value. `Address1` and `Address2` are the same because the Fragment portion is ignored for this comparison. The outcome is written to the console.

```
C#
// Create some Uris.
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
Uri address2 = new Uri("http://www.contoso.com/index.htm");
Uri address3 = new Uri("http://www.contoso.com/index.htm?date=today");
```

```
// The first two are equal because the fragment is ignored.
if (address1 == address2)
    Console.WriteLine("{0} is equal to {1}", address1.ToString(),
address2.ToString());

// The second two are not equal.
if (address2 != address3)
    Console.WriteLine("{0} is not equal to {1}", address2.ToString(),
address3.ToString());
```

Remarks

This overload uses the Equals method to determine whether the two Uri instances are equivalent. UserInfo and Fragment content is ignored when making this comparison.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.Inequality(Uri, Uri) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two Uri instances do not have the same value.

```
C#
public static bool operator != (Uri? uri1, Uri? uri2);
```

Parameters

uri1 Uri

A URI to compare with `uri2`.

uri2 Uri

A URI to compare with `uri1`.

Returns

Boolean

`true` if the two Uri instances are not equal; otherwise, `false`. If either parameter is `null`, this method returns `true`.

Examples

This example creates three Uri instances from strings and compares them to determine whether they represent the same value. `Address2` and `Address3` are not the same because `Address3` contains a Query that is not found in `Address2`. The outcome is written to the console.

```
C#
// Create some Uris.
Uri address1 = new Uri("http://www.contoso.com/index.htm#search");
```

```
Uri address2 = new Uri("http://www.contoso.com/index.htm");
Uri address3 = new Uri("http://www.contoso.com/index.htm?date=today");

// The first two are equal because the fragment is ignored.
if (address1 == address2)
    Console.WriteLine("{0} is equal to {1}", address1.ToString(),
address2.ToString());

// The second two are not equal.
if (address2 != address3)
    Console.WriteLine("{0} is not equal to {1}", address2.ToString(),
address3.ToString());
```

Remarks

This overload uses the Equals method to determine whether the two Uri instances are not equivalent. UserInfo and Fragment content is ignored when making this comparison.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Uri.IFormattable.ToString(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Formats the value of the current instance using the specified format.

C#

```
string IFormattable.ToString (string format, IFormatProvider
formatProvider);
```

Parameters

`format` String

The format to use. -or- A `null` reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`formatProvider` IFormatProvider

The provider to use to format the value. -or- A `null` reference (`Nothing` in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance into the provided span of characters.

C#

```
bool ISpanFormattable.TryFormat (Span<char> destination, out int
charsWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Uri.ISerializable.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the data needed to serialize the current instance.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo serializationInfo,
System.Runtime.Serialization.StreamingContext streamingContext);
```

Parameters

`serializationInfo` SerializationInfo

The information required to serialize the Uri.

`streamingContext` StreamingContext

An object containing the source and destination of the serialized stream associated with the Uri.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides a custom constructor for uniform resource identifiers (URIs) and modifies URIs for the Uri class.

C#

```
public class UriBuilder
```

Inheritance Object → UriBuilder

Remarks

The UriBuilder class provides a convenient way to modify the contents of a Uri instance without creating a new Uri instance for each modification.

The UriBuilder properties provide read/write access to the read-only Uri properties so that they can be modified.

Constructors

Expand table

UriBuilder()	Initializes a new instance of the UriBuilder class.
UriBuilder(String)	Initializes a new instance of the UriBuilder class with the specified URI.
UriBuilder(String, String)	Initializes a new instance of the UriBuilder class with the specified scheme and host.
UriBuilder(String, String, Int32)	Initializes a new instance of the UriBuilder class with the specified scheme, host, and port.
UriBuilder(String, String,	Initializes a new instance of the UriBuilder class with the specified

Int32, String)	scheme, host, port number, and path.
UriBuilder(String, String, Int32, String, String)	Initializes a new instance of the UriBuilder class with the specified scheme, host, port number, path, and query string or fragment identifier.
UriBuilder(Uri)	Initializes a new instance of the UriBuilder class with the specified Uri instance.

Properties

Expand table

Fragment	Gets or sets the fragment portion of the URI, including the leading '#' character if not empty.
Host	Gets or sets the Domain Name System (DNS) host name or IP address of a server.
Password	Gets or sets the password associated with the user that accesses the URI.
Path	Gets or sets the path to the resource referenced by the URI.
Port	Gets or sets the port number of the URI.
Query	Gets or sets any query information included in the URI, including the leading '?' character if not empty.
Scheme	Gets or sets the scheme name of the URI.
Uri	Gets the Uri instance constructed by the specified UriBuilder instance.
User Name	Gets or sets the user name associated with the user that accesses the URI.

Methods

Expand table

Equals(Object)	Compares an existing Uri instance with the contents of the UriBuilder for equality.
GetHashCode()	Returns the hash code for the URI.
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise	Creates a shallow copy of the current Object.

Clone()	(Inherited from Object)
ToString()	Returns the display string for the specified UriBuilder instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Uri

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the UriBuilder class.

Overloads

Expand table

UriBuilder()	Initializes a new instance of the UriBuilder class.
UriBuilder(String)	Initializes a new instance of the UriBuilder class with the specified URI.
UriBuilder(Uri)	Initializes a new instance of the UriBuilder class with the specified Uri instance.
UriBuilder(String, String)	Initializes a new instance of the UriBuilder class with the specified scheme and host.
UriBuilder(String, String, Int32)	Initializes a new instance of the UriBuilder class with the specified scheme, host, and port.
UriBuilder(String, String, Int32, String)	Initializes a new instance of the UriBuilder class with the specified scheme, host, port number, and path.
UriBuilder(String, String, Int32, String, String)	Initializes a new instance of the UriBuilder class with the specified scheme, host, port number, path, and query string or fragment identifier.

UriBuilder()

Initializes a new instance of the UriBuilder class.

C#

```
public UriBuilder ();
```

Remarks

The parameterless constructor creates a new instance of the UriBuilder class with its properties initialized as follows.

Expand table

Property	Initial Value
Fragment	Empty
Host	"loopback"
Password	Empty
Path	"/"
Port	-1
Query	Empty
Scheme	"http"
UserName	Empty

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(String)

Initializes a new instance of the UriBuilder class with the specified URI.

```
C#
```

```
public UriBuilder (string uri);
```

Parameters

`uri` String

A URI string.

Exceptions

ArgumentNullException

`uri` is `null`.

UriFormatException

`uri` is a zero-length string or contains only spaces.

-or-

The parsing routine detected a scheme in an invalid form.

-or-

The parser detected more than two consecutive slashes in a URI that does not use the "file" scheme.

-or-

`uri` is not a valid URI.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

Remarks

This constructor initializes a new instance of the UriBuilder class with the Fragment, Host, Path, Port, Query, Scheme, and Uri properties set as specified in `uri`.

If `uri` does not specify a scheme, the scheme defaults to "http:".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(Uri)

Initializes a new instance of the UriBuilder class with the specified Uri instance.

C#

```
public UriBuilder (Uri uri);
```

Parameters

`uri` Uri

An instance of the Uri class.

Exceptions

ArgumentNullException

`uri` is `null`.

Remarks

This constructor initializes a new instance of the UriBuilder class with the Fragment, Host, Path, Port, Query, Scheme, and Uri properties set as specified in `uri`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(String, String)

Initializes a new instance of the UriBuilder class with the specified scheme and host.

```
C#

public UriBuilder (string? schemeName, string? hostName);
```

Parameters

`schemeName` String

An Internet access protocol.

`hostName` String

A DNS-style domain name or IP address.

Examples

The following example creates a UriBuilder instance that contains the URI `http://www.contoso.com/`.

```
C#

UriBuilder myUri = new UriBuilder("http","www.contoso.com");
```

Remarks

The UriBuilder instance is initialized with the Scheme property set to `schemeName` and the Host property set to `hostName`. Port is initialized to the value -1 to indicate the default port for the scheme should be used, and the Path property is set to the slash character (/).

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(String, String, Int32)

Initializes a new instance of the UriBuilder class with the specified scheme, host, and port.

C#

```
public UriBuilder (string? scheme, string? host, int portNumber);
```

Parameters

`scheme` String

An Internet access protocol.

`host` String

A DNS-style domain name or IP address.

`portNumber` Int32

An IP port number for the service.

Exceptions

ArgumentOutOfRangeException

`portNumber` is less than -1 or greater than 65,535.

Examples

The following example creates a UriBuilder instance that contains the URI `http://www.contoso.com:8080/`.

```
C#

UriBuilder myUri = new UriBuilder("http","www.contoso.com",8080);
```

Remarks

The UriBuilder instance is initialized with the Scheme property set to `schemeName`, the Host property set to `hostName`, and the Port property set to `portNumber`. The Path property is set to the slash character (/).

If the `portNumber` is set to a value of -1, this indicates that the default port value for the scheme will be used to connect to the host.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(String, String, Int32, String)

Initializes a new instance of the UriBuilder class with the specified scheme, host, port number, and path.

```
C#

public UriBuilder (string? scheme, string? host, int port, string?
pathValue);
```

Parameters

`scheme` String

An Internet access protocol.

`host` String

A DNS-style domain name or IP address.

`port` Int32

An IP port number for the service.

`pathValue` String

The path to the Internet resource.

Exceptions

ArgumentOutOfRangeException

`port` is less than -1 or greater than 65,535.

Examples

The following example creates a UriBuilder instance that contains the URI `http://www.contoso.com:8080/index.htm`.

```
C#

UriBuilder myUri = new
UriBuilder("http","www.contoso.com",8080,"index.htm");
```

Remarks

The UriBuilder instance is initialized with the Scheme property set to `schemeName`, the Host property set to `hostName`, the Port property set to `portNumber`, and the Path property set to `pathValue`.

If the `portNumber` is set to a value of -1, this indicates that the default port value for the scheme will be used to connect to the host.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriBuilder(String, String, Int32, String, String)

Initializes a new instance of the UriBuilder class with the specified scheme, host, port number, path, and query string or fragment identifier.

C#

```
public UriBuilder (string? scheme, string? host, int port, string? path,
string? extraValue);
```

Parameters

`scheme` String

An Internet access protocol.

`host` String

A DNS-style domain name or IP address.

`port` Int32

An IP port number for the service.

`path` String

The path to the Internet resource.

`extraValue` String

A query string or fragment identifier.

Exceptions

ArgumentException

`extraValue` is neither `null` nor Empty, nor does a valid fragment identifier begin with a number sign (#), nor a valid query string begin with a question mark (?).

ArgumentOutOfRangeException

`port` is less than -1 or greater than 65,535.

Examples

The following example creates a UriBuilder instance that contains the URI `http://www.contoso.com:8080/index.htm#top`.

```
C#

UriBuilder myUri = new
UriBuilder("http","www.contoso.com",8080,"index.htm","#top");
```

Remarks

The UriBuilder instance is initialized with the Scheme property set to `schemeName`, the Host property set to `hostName`, the Port property set to `portNumber`, and the Path property set to `pathValue`. If `extraValue` begins with a number sign (#), then Fragment is set to `extraValue`. If `extraValue` begins with a question mark (?), then Query is set to `extraValue`.

If the `portNumber` is set to a value of -1, this indicates that the default port value for the scheme will be used to connect to the host.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriBuilder.Fragment Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the fragment portion of the URI, including the leading '#' character if not empty.

```
C#

public string Fragment { get; set; }
```

Property Value

String

The fragment portion of the URI.

Examples

The following example creates the URI `"http://www.contoso.com/index.htm#main"`.

```
C#

UriBuilder uBuild = new UriBuilder("http://www.contoso.com/");
uBuild.Path = "index.htm";
uBuild.Fragment = "main";

Uri myUri = uBuild.Uri;
```

Remarks

The Fragment property contains any text following a fragment marker (#) in the URI, including the marker itself. When setting the Fragment property:

- In .NET Framework, the fragment marker is always prepended to the string, even if one is already present.

- In .NET 5 (and .NET Core) and later versions, the fragment marker is prepended to the string if it's not already present.

> **Note**
>
> To append a value to an existing fragment in .NET Framework, you must remove the leading fragment marker before setting the property with the new value. This is because .NET Framework always prepends the fragment marker when setting the property. .NET 5 (and .NET Core) and later versions are tolerant to a leading fragment marker, and will only prepend one if necessary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder.Host Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the Domain Name System (DNS) host name or IP address of a server.

C#

```
public string Host { get; set; }
```

Property Value

String

The DNS host name or IP address of the server.

Remarks

The Host property contains the fully qualified DNS host name or IP address of the server.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriBuilder.Password Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the password associated with the user that accesses the URI.

C#

```
public string Password { get; set; }
```

Property Value

String

The password of the user that accesses the URI.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UriBuilder.Path Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the path to the resource referenced by the URI.

```
C#
public string Path { get; set; }
```

Property Value

String

The path to the resource referenced by the URI.

Remarks

The Path property contains the path information that the server uses to resolve requests for information. Typically, this is the path to the desired information on the server's file system. It can also indicate the application or script that the server must run to provide the information.

The path information does not include the scheme, host name, or query portion of the URI. The Path property always returns at least a slash (/).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriBuilder.Port Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the port number of the URI.

```
C#
public int Port { get; set; }
```

Property Value

Int32

The port number of the URI.

Exceptions

ArgumentOutOfRangeException

The port cannot be set to a value less than -1 or greater than 65,535.

Remarks

The port number defines the protocol port for contacting the host referenced in the URI. If a port is not specified as part of the URI, the Port property returns the value of -1 to indicate that the default port value for the protocol scheme will be used to connect to the host. If the Port property is set to a value of -1, this indicates that the default port value for the protocol scheme will be used to connect to the host.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Uri
- Scheme

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder.Query Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets any query information included in the URI, including the leading '?' character if not empty.

```
C#

public string Query { get; set; }
```

Property Value

String

The query information included in the URI.

Examples

The following example sets the Query property.

```
C#

UriBuilder baseUri = new UriBuilder("http://www.contoso.com/default.aspx?
Param1=7890");
string queryToAppend = "param2=1234";

if (baseUri.Query != null && baseUri.Query.Length > 1)
    // Note: In .NET Core and .NET 5+, you can simplify by removing
    // the call to Substring(), which removes the leading "?" character.
    baseUri.Query = baseUri.Query.Substring(1) + "&" + queryToAppend;
else
    baseUri.Query = queryToAppend;
```

Remarks

The Query property contains any query information included in the URI. Query information is separated from the path information by a question mark (?) and continues to the end of the URI. The query information that's returned includes the leading question mark. When setting the Query property:

- In .NET Framework, a question mark is always prepended to the string, even if the string already starts with a question mark.
- In .NET 5 (and .NET Core) and later versions, a question mark is prepended to the string if it's not already present.

The query information is escaped according to RFC 2396.

> **Note**
>
> To append a value to existing query information in .NET Framework, you must remove the leading question mark before setting the property with the new value. This is because .NET Framework always prepends the question mark when setting the property. .NET 5 (and .NET Core) and later versions are tolerant to a leading question mark, and will only prepend one if necessary.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Uri
- Query

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriBuilder.Scheme Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the scheme name of the URI.

C#

```
public string Scheme { get; set; }
```

Property Value

String

The scheme of the URI.

Exceptions

ArgumentException

The scheme cannot be set to an invalid scheme name.

Remarks

The following table lists the valid scheme names for the Scheme property.

Expand table

Scheme	Description
file	The resource is a file on the local computer.
ftp	The resource is accessed through FTP.
gopher	The resource is accessed through the Gopher protocol.
http	The resource is accessed through HTTP.
https	The resource is accessed through SSL-encrypted HTTP.

Scheme	Description
mailto	The resource is an email address and is accessed through SMTP.
news	The resource is accessed through NNTP.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Uri
- Scheme

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder.Uri Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the Uri instance constructed by the specified UriBuilder instance.

C#

```
public Uri Uri { get; }
```

Property Value

Uri

The URI constructed by the UriBuilder.

Exceptions

UriFormatException

The URI constructed by the UriBuilder properties is invalid.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

Remarks

The Uri property contains the Uri created by the UriBuilder. Any changes made to the UriBuilder properties are reflected in the Uri property.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Uri

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder.UserName Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the user name associated with the user that accesses the URI.

C#

```
public string UserName { get; set; }
```

Property Value

String

The name of the user that accesses the URI.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UriBuilder.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares an existing Uri instance with the contents of the UriBuilder for equality.

C#

```
public override bool Equals (object? rparam);
```

Parameters

`rparam` Object

The object to compare with the current instance.

Returns

Boolean

`true` if `rparam` represents the same Uri as the Uri constructed by this UriBuilder instance; otherwise, `false`.

Remarks

The Equals method compares a specified Uri instance with the Uri instance built by the UriBuilder instance. If the two are the same, the Equals method returns `true`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriBuilder.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the URI.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code generated for the URI.

Remarks

The hash code is generated without including any fragment. The URIs `http://www.contoso.com/index.htm#search` and `http://www.contoso.com/index.htm` generate the same hash code.

The hash codes for Uri A and B are guaranteed to be the same when A.Equals(B) is `true`.

This method implements Object.GetHashCode.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriBuilder.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the display string for the specified UriBuilder instance.

C#

```
public override string ToString ();
```

Returns

String

The string that contains the unescaped display string of the UriBuilder.

Exceptions

UriFormatException

The UriBuilder instance has a bad password.

Note: In .NET for Windows Store apps or the Portable Class Library, catch the base class exception, FormatException, instead.

Remarks

The display string contains the Port property value if one of the following conditions is true:

- The Port property was set.
- The port was specified when the UriBuilder was constructed using one of the UriBuilder constructors that accept a port number parameter.

In either case, the display string does not contain the Port property value if the port number was set to a value of -1 (for the default port).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriComponents Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies the parts of a Uri.

This enumeration supports a bitwise combination of its member values.

C#

```
[System.Flags]
public enum UriComponents
```

Inheritance Object → ValueType → Enum → UriComponents

Attributes FlagsAttribute

Fields

Expand table

AbsoluteUri	127	The Scheme, UserInfo, Host, Port, LocalPath, Query, and Fragment data.
Fragment	64	The Fragment data.
Host	4	The Host data.
HostAndPort	132	The Host and Port data. If no port data is in the Uri and a default port has been assigned to the Scheme, the default port is returned. If there is no default port, -1 is returned.
HttpRequestUrl	61	The Scheme, Host, Port, LocalPath, and Query data.
KeepDelimiter	1073741824	Specifies that the delimiter should be included.
NormalizedHost	256	The normalized form of the Host.
Path	16	The LocalPath data.

PathAndQuery	48	The LocalPath and Query data. Also see PathAndQuery.
Port	8	The Port data.
Query	32	The Query data.
Scheme	1	The Scheme data.
SchemeAndServer	13	The Scheme, Host, and Port data.
SerializationInfoString	-2147483648	The complete Uri context that is needed for Uri Serializers. The context includes the IPv6 scope.
StrongAuthority	134	The UserInfo, Host, and Port data. If no port data is in the Uri and a default port has been assigned to the Scheme, the default port is returned. If there is no default port, -1 is returned.
StrongPort	128	The Port data. If no port data is in the Uri and a default port has been assigned to the Scheme, the default port is returned. If there is no default port, -1 is returned.
UserInfo	2	The UserInfo data.

Remarks

This enumeration is used to identify the parts of a Uri. This enumeration is used with the Uri.GetComponents method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UriCreationOptions Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides options that control how a Uri is created and behaves.

C#

```
public struct UriCreationOptions
```

Inheritance Object → ValueType → UriCreationOptions

Properties

Expand table

DangerousDisablePathAndQueryCanonicalization	Gets or sets a value that indicates whether the path and query are validated and normalized.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

UriCreationOptions.DangerousDisablePathAndQueryCanonicalization Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets a value that indicates whether the path and query are validated and normalized.

C#

```
public bool DangerousDisablePathAndQueryCanonicalization { get; set; }
```

Property Value

Boolean

`true` to disable path and query validation; `false` to enable it.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriFormat Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Controls how URI information is escaped.

```
C#

public enum UriFormat
```

Inheritance Object → ValueType → Enum → UriFormat

Fields

Expand table

SafeUnescaped	3	Characters that have a reserved meaning in the requested URI components remain escaped. All others are not escaped.
Unescaped	2	No escaping is performed.
UriEscaped	1	Escaping is performed according to the rules in RFC 2396.

Remarks

This enumeration is used by the GetComponents method to specify the level of character escaping in the returned URI.

RFC 2396 defines the standard escaping for URIs, and is available at https://rfc-editor.org.

SafeUnescaped leaves the following characters escaped if the character has a reserved meaning in the requested UriComponents: "%", "#", "?", "/", "\", and "@".

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriFormatException Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

The exception that is thrown when an invalid Uniform Resource Identifier (URI) is detected.

```
C#
public class UriFormatException : FormatException
```

Inheritance Object → Exception → SystemException → FormatException → UriFormatException

Implements ISerializable

Remarks

The UriFormatException is thrown by the Uri class constructor if the supplied URI could not be correctly parsed. The format for a valid URI is defined in RFC 2396.

The text of the Message property indicates the reason that the exception was thrown.

> **Note**
>
> This exception is not included in the **.NET for Windows Store apps** or the **Portable Class Library**, but it is thrown by some members that are. To catch the exception in that case, write a `catch` statement for **FormatException** instead.

Constructors

Expand table

UriFormatException()	Initializes a new instance of the UriFormatException class.

UriFormatException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the UriFormatException class from the specified SerializationInfo and StreamingContext instances.
UriFormatException(String)	Initializes a new instance of the UriFormatException class with the specified message.
UriFormatException(String, Exception)	Initializes a new instance of the UriFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Properties

Expand table

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception. (Inherited from Exception)
HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception. (Inherited from Exception)
Inner Exception	Gets the Exception instance that caused the current exception. (Inherited from Exception)
Message	Gets a message that describes the current exception. (Inherited from Exception)
Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
StackTrace	Gets a string representation of the immediate frames on the call stack. (Inherited from Exception)
TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetBaseException()	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** When overridden in a derived class, sets the SerializationInfo with information about the exception. (Inherited from Exception)
GetType()	Gets the runtime type of the current instance. (Inherited from Exception)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Creates and returns a string representation of the current exception. (Inherited from Exception)

Events

Expand table

SerializeObjectState	**Obsolete.** Occurs when an exception is serialized to create an exception state object that contains serialized data about the exception. (Inherited from Exception)

Explicit Interface Implementations

Expand table

ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo instance with the data that is needed to serialize the UriFormatException.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriFormatException Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the UriFormatException class.

Overloads

Expand table

UriFormatException()	Initializes a new instance of the UriFormatException class.
UriFormatException(String)	Initializes a new instance of the UriFormatException class with the specified message.
UriFormatException(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the UriFormatException class from the specified SerializationInfo and StreamingContext instances.
UriFormatException(String, Exception)	Initializes a new instance of the UriFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

UriFormatException()

Initializes a new instance of the UriFormatException class.

C#

```
public UriFormatException ();
```

Remarks

The parameterless constructor initializes a new instance of the UriFormatException class with all fields set to `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriFormatException(String)

Initializes a new instance of the UriFormatException class with the specified message.

C#

```
public UriFormatException (string? textString);
```

Parameters

`textString` String

The error message string.

Remarks

The UriFormatException constructor initializes the UriFormatException instance with the Message property set to the value of `textString`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

UriFormatException(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the UriFormatException class from the specified SerializationInfo and StreamingContext instances.

C#

```
[System.Obsolete("This API supports obsolete formatter-based
serialization. It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-
warnings/{0}")]
protected UriFormatException
(System.Runtime.Serialization.SerializationInfo serializationInfo,
System.Runtime.Serialization.StreamingContext streamingContext);
```

Parameters

`serializationInfo` SerializationInfo

A SerializationInfo that contains the information that is required to serialize the new UriFormatException.

`streamingContext` StreamingContext

A StreamingContext that contains the source of the serialized stream that is associated with the new UriFormatException.

Attributes ObsoleteAttribute

Remarks

This constructor implements the ISerializable interface for the UriFormatException class.

See also

- ISerializable

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

UriFormatException(String, Exception)

Initializes a new instance of the UriFormatException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C#

```
public UriFormatException (string? textString, Exception? e);
```

Parameters

`textString` String

The message that describes the exception. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

`e` Exception

The exception that is the cause of the current exception. If the `innerException` parameter is not `null`, the current exception is raised in a `catch` block that handles the inner exception.

Remarks

An exception that is thrown as a direct result of a previous exception should include a reference to the previous exception in the InnerException property. The InnerException property returns the same value that is passed into the constructor, or `null` if the InnerException property does not supply the inner exception value to the constructor.

The following table shows the initial property values for an instance of UriFormatException.

Expand table

Property	Value
InnerException	`null`.
Message	The error message string specified in `message`.

See also

- Handling and Throwing Exceptions

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriFormatException.ISerializable.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Populates a SerializationInfo instance with the data that is needed to serialize the UriFormatException.

C#

```
void ISerializable.GetObjectData
(System.Runtime.Serialization.SerializationInfo serializationInfo,
System.Runtime.Serialization.StreamingContext streamingContext);
```

Parameters

`serializationInfo` SerializationInfo

A SerializationInfo that will hold the serialized data for the UriFormatException.

`streamingContext` StreamingContext

A StreamingContext that contains the destination of the serialized stream that is associated with the new UriFormatException.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ISerializable

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriHostNameType Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines host name types for the CheckHostName(String) method.

C#

```
public enum UriHostNameType
```

Inheritance Object → ValueType → Enum → UriHostNameType

Fields

Expand table

Basic	1	The host is set, but the type cannot be determined.
Dns	2	The host name is a domain name system (DNS) style host name.
IPv4	3	The host name is an Internet Protocol (IP) version 4 host address.
IPv6	4	The host name is an Internet Protocol (IP) version 6 host address.
Unknown	0	The type of the host name is not supplied.

Remarks

The UriHostNameType enumeration defines the values that the Uri.CheckHostName method can return.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriKind Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines the different kinds of URIs.

C#

```
public enum UriKind
```

Inheritance Object → ValueType → Enum → UriKind

Fields

Expand table

Absolute	1	The URI is absolute.
Relative	2	The URI is relative.
RelativeOrAbsolute	0	The URI kind is indeterminate.

Remarks

Absolute URIs are characterized by a complete reference to the resource (example: `http://www.contoso.com/index.html`), while a relative URI depends on a previously defined base URI (example: `/index.html`).

The following list shows some APIs that use the `UriKind` enum:

- Uri.Uri(String, UriKind)
- Uri.IsWellFormedUriString(String, UriKind)
- Uri.TryCreate(String, UriKind, Uri)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Parses a new URI scheme. This is an abstract class.

```
C#
public abstract class UriParser
```

Inheritance Object → UriParser

Derived System.FileStyleUriParser
System.FtpStyleUriParser
System.GenericUriParser
System.GopherStyleUriParser
System.HttpStyleUriParser
More...

Remarks

The UriParser class enables you to create parsers for new URI schemes. You can write these parsers in their entirety, or the parsers can be derived from well-known schemes (HTTP, FTP, and other schemes based on network protocols). If you want to create a completely new parser, inherit from GenericUriParser. If you want to create a parser that extends a well-known URI scheme, inherit from FtpStyleUriParser, HttpStyleUriParser, FileStyleUriParser, GopherStyleUriParser, or LdapStyleUriParser.

Microsoft strongly recommends that you use a parser shipped with .NET. Building your own parser increases the complexity of your application, and will not perform as well as the shipped parsers.

Constructors

Expand table

UriParser()	Constructs a default URI parser.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetComponents(Uri, UriComponents, UriFormat)	Gets the components from a URI.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
InitializeAndValidate(Uri, UriFormatException)	Initialize the state of the parser and validate the URI.
IsBaseOf(Uri, Uri)	Determines whether `baseUri` is a base URI for `relativeUri`.
IsKnownScheme(String)	Indicates whether the parser for a scheme is registered.
IsWellFormedOriginalString(Uri)	Indicates whether a URI is well-formed.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
OnNewUri()	Invoked by a Uri constructor to get a UriParser instance.
OnRegister(String, Int32)	Invoked by the Framework when a UriParser method is registered.
Register(UriParser, String, Int32)	Associates a scheme and port number with a UriParser.
Resolve(Uri, Uri, UriFormatException)	Called by Uri constructors and TryCreate to resolve a relative URI.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Constructs a default URI parser.

C#

```
protected UriParser ();
```

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser.GetComponents(Uri, UriComponents, UriFormat) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the components from a URI.

```
C#

protected virtual string GetComponents (Uri uri, UriComponents components,
UriFormat format);
```

Parameters

`uri` Uri

The URI to parse.

`components` UriComponents

The UriComponents to retrieve from `uri`.

`format` UriFormat

One of the UriFormat values that controls how special characters are escaped.

Returns

String

A string that contains the components.

Exceptions

ArgumentOutOfRangeException

`uriFormat` is invalid.

-or-

`uriComponents` is not a combination of valid UriComponents values.

InvalidOperationException
`uri` requires user-driven parsing

-or-

`uri` is not an absolute URI. Relative URIs cannot be used with this method.

Remarks

Use the GetComponents method to determine the value of various parts of the URI, such as the Scheme, Host, or Port.

The components are returned in the order that they appear in the URI. For example, if Scheme is specified, it appears first.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- UriComponents
- Host
- Port
- Scheme
- UriFormat
- GetComponents(Uri, UriComponents, UriFormat)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UriParser.InitializeAndValidate(Uri, UriFormatException) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initialize the state of the parser and validate the URI.

C#

```
protected virtual void InitializeAndValidate (Uri uri, out
UriFormatException? parsingError);
```

Parameters

`uri` Uri

The Uri to validate.

`parsingError` UriFormatException

When this method returns, contains the validation errors, if any.

Remarks

The InitializeAndValidate method is called every time a Uri is instantiated.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriParser.IsBaseOf(Uri, Uri) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether `baseUri` is a base URI for `relativeUri`.

C#

```
protected virtual bool IsBaseOf (Uri baseUri, Uri relativeUri);
```

Parameters

`baseUri` Uri

The base URI.

`relativeUri` Uri

The URI to test.

Returns

Boolean

`true` if `baseUri` is a base URI for `relativeUri`; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriParser.IsKnownScheme(String) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether the parser for a scheme is registered.

C#

```
public static bool IsKnownScheme (string schemeName);
```

Parameters

schemeName String

The scheme name to check.

Returns

Boolean

`true` if `schemeName` has been registered; otherwise, `false`.

Exceptions

ArgumentNullException

The `schemeName` parameter is null.

ArgumentOutOfRangeException

The `schemeName` parameter is not valid.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser.IsWellFormedOriginalString(Uri) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether a URI is well-formed.

```
C#
protected virtual bool IsWellFormedOriginalString (Uri uri);
```

Parameters

`uri` Uri

The URI to check.

Returns

Boolean

`true` if `uri` is well-formed; otherwise, `false`.

Remarks

The IsWellFormedOriginalString method will return false if:

- The string is not correctly escaped per RFC 2396.
- The string represents an absolute URI that references an implicit file URI.
- The string represents an absolute URI that is missing a forward slash before the path component.
- The string contains unescaped backslashes, even if they will be treated as forward slashes.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser.OnNewUri Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Invoked by a Uri constructor to get a UriParser instance.

C#

```
protected virtual UriParser OnNewUri ();
```

Returns

UriParser

A UriParser for the constructed Uri.

Remarks

The default implementation returns the same UriParser instance. You may need to override this method and return a different UriParser for each invocation if you are operating in a multithreaded environment.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriParser.OnRegister(String, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Invoked by the Framework when a UriParser method is registered.

C#

```
protected virtual void OnRegister (string schemeName, int defaultPort);
```

Parameters

`schemeName` String

The scheme that is associated with this UriParser.

`defaultPort` Int32

The port number of the scheme.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

UriParser.Register(UriParser, String, Int32) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Associates a scheme and port number with a UriParser.

```
C#

public static void Register (UriParser uriParser, string schemeName, int
defaultPort);
```

Parameters

`uriParser` UriParser

The URI parser to register.

`schemeName` String

The name of the scheme that is associated with this parser.

`defaultPort` Int32

The default port number for the specified scheme.

Exceptions

ArgumentNullException

`uriParser` parameter is null

-or-

`schemeName` parameter is null.

ArgumentOutOfRangeException

`schemeName` parameter is not valid

-or-

`defaultPort` parameter is not valid. The `defaultPort` parameter is less than -1 or greater than 65,534.

Remarks

If the `defaultPort` parameter is set to -1, the Register method registers no default value for the port number.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriParser.Resolve(Uri, Uri, UriFormatException) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Called by Uri constructors and TryCreate to resolve a relative URI.

C#

```
protected virtual string? Resolve (Uri baseUri, Uri? relativeUri, out
UriFormatException? parsingError);
```

Parameters

`baseUri` Uri

A base URI.

`relativeUri` Uri

A relative URI.

`parsingError` UriFormatException

When this method returns, contains the errors that occurred during the resolve process, if any.

Returns

String

The string of the resolved relative Uri.

Exceptions

InvalidOperationException

`baseUri` parameter is not an absolute Uri

-or-

`baseUri` parameter requires user-driven parsing.

Remarks

Uri constructors and Uri.TryCreate use Resolve to construct a URI from `baseUri` and `relativeUri`.

If a parsing error occurs, the returned string for the resolved relative Uri is null.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriPartial Enum

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Defines the parts of a URI for the GetLeftPart(UriPartial) method.

C#

```
public enum UriPartial
```

Inheritance Object → ValueType → Enum → UriPartial

Fields

Expand table

Authority	1	The scheme and authority segments of the URI.
Path	2	The scheme, authority, and path segments of the URI.
Query	3	The scheme, authority, path, and query segments of the URI.
Scheme	0	The scheme segment of the URI.

Remarks

The UriPartial enumeration defines the values that can be passed to the Uri.GetLeftPart method.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter Class

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Converts a String type to a Uri type, and vice versa.

C#

```
public class UriTypeConverter : System.ComponentModel.TypeConverter
```

Inheritance Object → TypeConverter → UriTypeConverter

Constructors

Expand table

UriTypeConverter()	Initializes a new instance of the UriTypeConverter class.

Methods

Expand table

CanConvertFrom(ITypeDescriptorContext, Type)	Returns whether this converter can convert an object of the given type to the type of this converter.
CanConvertFrom(Type)	Returns whether this converter can convert an object of the given type to the type of this converter. (Inherited from TypeConverter)
CanConvertTo(ITypeDescriptorContext, Type)	Returns whether this converter can convert the object to the specified type, using the specified context.
CanConvertTo(Type)	Returns whether this converter can convert the object to the specified type. (Inherited from TypeConverter)

ConvertFrom(ITypeDescriptorContext, CultureInfo, Object)	Converts the given object to the type of this converter, using the specified context and culture information.
ConvertFrom(Object)	Converts the given value to the type of this converter. (Inherited from TypeConverter)
ConvertFromInvariantString(ITypeDescriptorContext, String)	Converts the given string to the type of this converter, using the invariant culture and the specified context. (Inherited from TypeConverter)
ConvertFromInvariantString(String)	Converts the given string to the type of this converter, using the invariant culture. (Inherited from TypeConverter)
ConvertFromString(ITypeDescriptorContext, CultureInfo, String)	Converts the given text to an object, using the specified context and culture information. (Inherited from TypeConverter)
ConvertFromString(ITypeDescriptorContext, String)	Converts the given text to an object, using the specified context. (Inherited from TypeConverter)
ConvertFromString(String)	Converts the specified text to an object. (Inherited from TypeConverter)
ConvertTo(ITypeDescriptorContext, CultureInfo, Object, Type)	Converts a given value object to the specified type, using the specified context and culture information.
ConvertTo(Object, Type)	Converts the given value object to the specified type, using the arguments. (Inherited from TypeConverter)
ConvertToInvariantString(ITypeDescriptorContext, Object)	Converts the specified value to a culture-invariant string representation, using the specified context. (Inherited from TypeConverter)
ConvertToInvariantString(Object)	Converts the specified value to a culture-invariant string representation. (Inherited from TypeConverter)
ConvertToString(ITypeDescriptorContext, CultureInfo, Object)	Converts the given value to a string representation, using the specified context and culture information. (Inherited from TypeConverter)
ConvertToString(ITypeDescriptorContext, Object)	Converts the given value to a string representation, using the given context. (Inherited from TypeConverter)
ConvertToString(Object)	Converts the specified value to a string representation. (Inherited from TypeConverter)

CreateInstance(IDictionary)	Re-creates an Object given a set of property values for the object. (Inherited from TypeConverter)
CreateInstance(ITypeDescriptorContext, IDictionary)	Creates an instance of the type that this TypeConverter is associated with, using the specified context, given a set of property values for the object. (Inherited from TypeConverter)
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetConvertFromException(Object)	Returns an exception to throw when a conversion cannot be performed. (Inherited from TypeConverter)
GetConvertToException(Object, Type)	Returns an exception to throw when a conversion cannot be performed. (Inherited from TypeConverter)
GetCreateInstanceSupported()	Returns whether changing a value on this object requires a call to the CreateInstance(IDictionary) method to create a new value. (Inherited from TypeConverter)
GetCreateInstanceSupported(ITypeDescriptorContext)	Returns whether changing a value on this object requires a call to CreateInstance(IDictionary) to create a new value, using the specified context. (Inherited from TypeConverter)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetProperties(ITypeDescriptorContext, Object)	Returns a collection of properties for the type of array specified by the value parameter, using the specified context. (Inherited from TypeConverter)
GetProperties(ITypeDescriptorContext, Object, Attribute[])	Returns a collection of properties for the type of array specified by the value parameter, using the specified context and attributes. (Inherited from TypeConverter)
GetProperties(Object)	Returns a collection of properties for the type of array specified by the value parameter. (Inherited from TypeConverter)
GetPropertiesSupported()	Returns whether this object supports properties. (Inherited from TypeConverter)

GetPropertiesSupported(ITypeDescriptorContext)	Returns whether this object supports properties, using the specified context. (Inherited from TypeConverter)
GetStandardValues()	Returns a collection of standard values from the default context for the data type this type converter is designed for. (Inherited from TypeConverter)
GetStandardValues(ITypeDescriptorContext)	Returns a collection of standard values for the data type this type converter is designed for when provided with a format context. (Inherited from TypeConverter)
GetStandardValuesExclusive()	Returns whether the collection of standard values returned from GetStandardValues() is an exclusive list. (Inherited from TypeConverter)
GetStandardValuesExclusive(ITypeDescriptorContext)	Returns whether the collection of standard values returned from GetStandardValues() is an exclusive list of possible values, using the specified context. (Inherited from TypeConverter)
GetStandardValuesSupported()	Returns whether this object supports a standard set of values that can be picked from a list. (Inherited from TypeConverter)
GetStandardValuesSupported(ITypeDescriptorContext)	Returns whether this object supports a standard set of values that can be picked from a list, using the specified context. (Inherited from TypeConverter)
GetType()	Gets the Type of the current instance. (Inherited from Object)
IsValid(ITypeDescriptorContext, Object)	Returns whether the given value object is a Uri or a Uri can be created from it.
IsValid(Object)	Returns whether the given value object is valid for this type. (Inherited from TypeConverter)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
SortProperties(PropertyDescriptorCollection, String[])	Sorts a collection of properties. (Inherited from TypeConverter)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter Constructor

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Initializes a new instance of the UriTypeConverter class.

C#

```
public UriTypeConverter ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter.CanConvertFrom(ITypeDescriptorContext, Type) Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Returns whether this converter can convert an object of the given type to the type of this converter.

C#

```
public override bool CanConvertFrom
(System.ComponentModel.ITypeDescriptorContext? context, Type sourceType);
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type that you want to convert from.

Returns

Boolean

`true` if `sourceType` is a String type or a Uri type can be assigned from `sourceType`; otherwise, `false`.

Exceptions

ArgumentNullException

The `sourceType` parameter is `null`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter.CanConvertTo(ITypeDescriptorContext, Type) Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Returns whether this converter can convert the object to the specified type, using the specified context.

C#

```
public override bool CanConvertTo
(System.ComponentModel.ITypeDescriptorContext? context, Type?
destinationType);
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type that you want to convert to.

Returns

Boolean

`true` if `destinationType` is of type InstanceDescriptor, String, or Uri; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter.ConvertFrom Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Converts the given object to the type of this converter, using the specified context and culture information.

C#

```
public override object? ConvertFrom
(System.ComponentModel.ITypeDescriptorContext? context,
System.Globalization.CultureInfo? culture, object value);
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` Object

The Object to convert.

Returns

Object

An Object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

UriTypeConverter.ConvertTo Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Converts a given value object to the specified type, using the specified context and culture information.

```
C#

public override object ConvertTo
(System.ComponentModel.ITypeDescriptorContext? context,
System.Globalization.CultureInfo? culture, object? value, Type
destinationType);
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If `null` is passed, the current culture is assumed.

`value` Object

The Object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

Object

An Object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is `null`.

NotSupportedException

The conversion cannot be performed.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

UriTypeConverter.IsValid(ITypeDescriptorContext, Object) Method

Reference

Definition

Namespace: System

Assembly: System.ComponentModel.TypeConverter.dll

Returns whether the given value object is a Uri or a Uri can be created from it.

```
C#

public override bool IsValid (System.ComponentModel.ITypeDescriptorContext?
context, object? value);
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`value` Object

The Object to test for validity.

Returns

Boolean

`true` if `value` is a Uri or a String from which a Uri can be created; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides static methods for creating value tuples.

C#

```
public struct ValueTuple : IComparable, IComparable<ValueTuple>,
IEquatable<ValueTuple>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Inheritance Object → ValueType → ValueTuple

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple> , IEquatable<ValueTuple> , ITuple

Remarks

For more information about this API, see Value tuples.

Methods

Expand table

CompareTo(ValueTuple)	Compares the current ValueTuple instance to a specified ValueTuple instance.
Create()	Creates a new value tuple with zero components.
Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Creates a new value tuple with 8 components (an octuple).
Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Creates a new value tuple with 7 components (a septuple).

Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Creates a new value tuple with 6 components (a sexuple).
Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Creates a new value tuple with 5 components (a quintuple).
Create<T1,T2,T3,T4>(T1, T2, T3, T4)	Creates a new value tuple with 4 components (a quadruple).
Create<T1,T2,T3>(T1, T2, T3)	Creates a new value tuple with 3 components (a triple).
Create<T1,T2>(T1, T2)	Creates a new value tuple with 2 components (a pair).
Create<T1>(T1)	Creates a new value tuple with 1 component (a singleton).
Equals(Object)	Returns a value that indicates whether the current ValueTuple instance is equal to a specified object.
Equals(ValueTuple)	Determines whether two ValueTuple instances are equal. This method always returns `true`.
GetHashCode()	Returns the hash code for the current ValueTuple instance.
ToString()	Returns the string representation of this ValueTuple instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares this ValueTuple instance with a specified object and returns an indication of their relative values.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple instance to a specified object.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Returns the hash code for this ValueTuple instance.

ITuple.Item[Int32]	Returns an IndexOutOfRangeException. There are no elements in a `ValueTuple`.
ITuple.Length	Gets the length of this `ValueTuple` instance, which is always 0. There are no elements in a `ValueTuple`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.CompareTo(ValueTuple) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple instance to a specified ValueTuple instance.

```
C#
public int CompareTo (ValueTuple other);
```

Parameters

`other` ValueTuple

The object to compare with the current instance.

Returns

Int32

This method always returns 0.

Implements

CompareTo(T)

Exceptions

ArgumentException

`other` is not a ValueTuple instance.

Remarks

Because a ValueTuple instance has no elements, any two ValueTuple instances are considered to be equivalent.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.Create Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Creates a new value tuple instance.

Overloads

Expand table

Create()	Creates a new value tuple with zero components.
Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)	Creates a new value tuple with 8 components (an octuple).
Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Creates a new value tuple with 7 components (a septuple).
Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Creates a new value tuple with 6 components (a sexuple).
Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Creates a new value tuple with 5 components (a quintuple).
Create<T1,T2,T3,T4>(T1, T2, T3, T4)	Creates a new value tuple with 4 components (a quadruple).
Create<T1,T2,T3>(T1, T2, T3)	Creates a new value tuple with 3 components (a triple).
Create<T1,T2>(T1, T2)	Creates a new value tuple with 2 components (a pair).
Create<T1>(T1)	Creates a new value tuple with 1 component (a singleton).

Create()

Creates a new value tuple with zero components.

```
C#

public static ValueTuple Create ();
```

Returns

ValueTuple

A new value tuple with no components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)

Creates a new value tuple with 8 components (an octuple).

```
C#

public static (T1, T2, T3, T4, T5, T6, T7, (T8))
Create<T1,T2,T3,T4,T5,T6,T7,T8> (T1 item1, T2 item2, T3 item3, T4 item4,
T5 item5, T6 item6, T7 item7, T8 item8);
```

Type Parameters

T1

The type of the value tuple's first component.

`T2`

The type of the value tuple's second component.

`T3`

The type of the value tuple's third component.

`T4`

The type of the value tuple's fourth component.

`T5`

The type of the value tuple's fifth component.

`T6`

The type of the value tuple's sixth component.

`T7`

The type of the value tuple's seventh component.

`T8`

The type of the value tuple's eighth component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

`item4` T4

The value of the value tuple's fourth component.

`item5` T5

The value of the value tuple's fifth component.

`item6` T6

The value of the value tuple's sixth component.

`item7` T7

The value of the value tuple's seventh component.

`item8` T8

The value of the value tuple's eighth component.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7,ValueTuple<T8>>

A value tuple with 8 components.

Remarks

Create is a helper method that you can call to instantiate an 8-component value tuple without having to explicitly specify the types of its components.

> ⓘ **Note**
>
> You must call the **ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>** constructor to create a tuple with nine or more components unless your language provides a special syntax for this purpose. The static methods of the **ValueTuple** structure cannot be used to create a tuple with nine or more components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)

Creates a new value tuple with 7 components (a septuple).

```
C#

public static (T1, T2, T3, T4, T5, T6, T7) Create<T1,T2,T3,T4,T5,T6,T7>
(T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6, T7 item7);
```

Type Parameters

T1

The type of the value tuple's first component.

T2

The type of the value tuple's second component.

T3

The type of the value tuple's third component.

T4

The type of the value tuple's fourth component.

T5

The type of the value tuple's fifth component.

T6

The type of the value tuple's sixth component.

T7

The type of the value tuple's seventh component.

Parameters

item1 T1

The value of the value tuple's first component.

item2 T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

`item4` T4

The value of the value tuple's fourth component.

`item5` T5

The value of the value tuple's fifth component.

`item6` T6

The value of the value tuple's sixth component.

`item7` T7

The value of the value tuple's seventh component.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6,T7>

A value tuple with 7 components.

Remarks

Create is a helper method that you can call to instantiate an 7-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)

Creates a new value tuple with 6 components (a sexuple).

```
C#

public static (T1, T2, T3, T4, T5, T6) Create<T1,T2,T3,T4,T5,T6> (T1
item1, T2 item2, T3 item3, T4 item4, T5 item5, T6 item6);
```

Type Parameters

`T1`

The type of the value tuple's first component.

`T2`

The type of the value tuple's second component.

`T3`

The type of the value tuple's third component.

`T4`

The type of the value tuple's fourth component.

`T5`

The type of the value tuple's fifth component.

`T6`

The type of the value tuple's sixth component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

`item4` T4

The value of the value tuple's fourth component.

`item5` T5

The value of the value tuple's fifth component.

`item6` T6

The value of the value tuple's sixth component.

Returns

ValueTuple<T1,T2,T3,T4,T5,T6>

A value tuple with 6 components.

Remarks

Create is a helper method that you can call to instantiate a 6-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)

Creates a new value tuple with 5 components (a quintuple).

C#

```
public static (T1, T2, T3, T4, T5) Create<T1,T2,T3,T4,T5> (T1 item1, T2
item2, T3 item3, T4 item4, T5 item5);
```

Type Parameters

`T1`

The type of the value tuple's first component.

`T2`

The type of the value tuple's second component.

`T3`

The type of the value tuple's third component.

`T4`

The type of the value tuple's fourth component.

`T5`

The type of the value tuple's fifth component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

`item4` T4

The value of the value tuple's fourth component.

`item5` T5

The value of the value tuple's fifth component.

Returns

ValueTuple<T1,T2,T3,T4,T5>

A value tuple with 5 components.

Remarks

Create is a helper method that you can call to instantiate a 5-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3,T4>(T1, T2, T3, T4)

Creates a new value tuple with 4 components (a quadruple).

C#

```
public static (T1, T2, T3, T4) Create<T1,T2,T3,T4> (T1 item1, T2 item2,
T3 item3, T4 item4);
```

Type Parameters

T1

The type of the value tuple's first component.

T2

The type of the value tuple's second component.

T3

The type of the value tuple's third component.

T4

The type of the value tuple's fourth component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

`item4` T4

The value of the value tuple's fourth component.

Returns

ValueTuple<T1,T2,T3,T4>

A value tuple with 4 components.

Remarks

Create is a helper method that you can call to instantiate an 4-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2,T3>(T1, T2, T3)

Creates a new value tuple with 3 components (a triple).

```
C#
```

```
public static (T1, T2, T3) Create<T1,T2,T3> (T1 item1, T2 item2, T3
item3);
```

Type Parameters

`T1`

The type of the value tuple's first component.

`T2`

The type of the value tuple's second component.

`T3`

The type of the value tuple's third component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

`item3` T3

The value of the value tuple's third component.

Returns

ValueTuple<T1,T2,T3>

A value tuple with 3 components.

Remarks

Create is a helper method that you can call to instantiate an 3-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1,T2>(T1, T2)

Creates a new value tuple with 2 components (a pair).

```
C#
public static (T1, T2) Create<T1,T2> (T1 item1, T2 item2);
```

Type Parameters

`T1`

The type of the value tuple's first component.

`T2`

The type of the value tuple's second component.

Parameters

`item1` T1

The value of the value tuple's first component.

`item2` T2

The value of the value tuple's second component.

Returns

ValueTuple<T1,T2>

A value tuple with 2 components.

Remarks

Create is a helper method that you can call to instantiate an 2-component value tuple without having to explicitly specify the types of its components.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Create<T1>(T1)

Creates a new value tuple with 1 component (a singleton).

C#

```
public static (T1) Create<T1> (T1 item1);
```

Type Parameters

`T1`

The type of the value tuple's only component.

Parameters

`item1` T1

The value of the value tuple's only component.

Returns

ValueTuple<T1>

A value tuple with 1 component.

Remarks

Create is a helper method that you can call to instantiate a 1-component value tuple without having to explicitly specify the type of its component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple instance is equal to the specified object.

Overloads

Expand table

Equals(ValueTuple)	Determines whether two ValueTuple instances are equal. This method always returns `true`.
Equals(Object)	Returns a value that indicates whether the current ValueTuple instance is equal to a specified object.

Equals(ValueTuple)

Determines whether two ValueTuple instances are equal. This method always returns `true`.

```
C#

public bool Equals (ValueTuple other);
```

Parameters

`other` ValueTuple

The value tuple to compare with the current instance.

Returns

Boolean

This method always returns `true`.

Implements

Equals(T)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(Object)

Returns a value that indicates whether the current ValueTuple instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare to the current instance.

Returns

Boolean

`true` if `obj` is a ValueTuple instance; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for the current ValueTuple instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

This method always return 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the string representation of this ValueTuple instance.

C#

```
public override string ToString ();
```

Returns

String

This method always returns "()".

Remarks

The string returned by this method takes the form `()`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ValueTuple.IStructuralComparable.CompareTo(Object, IComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple instance to a specified object.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison. This parameter is ignored.

Returns

Int32

Returns 0 if `other` is a ValueTuple instance and 1 if `other` is `null`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a ValueTuple instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.IStructuralEquatable.Equals(Object, IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Remarks

This member is an explicit interface implementation. It can only be used when the ValueTuplet> instance is cast to an IStructuralEquatable interface. The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast to a ValueTuple object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- IEqualityComparer

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.IStructuralEquatable.GetHashCode(IEqualityComparer) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this ValueTuple instance.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

comparer IEqualityComparer

An object whose GetHashCode(Object) method computes the hash code. This parameter is ignored.

Returns

Int32

The hash code for this ValueTuple instance.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares this ValueTuple instance with a specified object and returns an indication of their relative values.

```
C#

int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

0 if `other` is a ValueTuple instance; otherwise, 1 if `other` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`other` is not a ValueTuple instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns an IndexOutOfRangeException. There are no elements in a `ValueTuple`.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

There is no acceptable value for `index`.

Property Value

Object

An IndexOutOfRangeException.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

There is no acceptable value for `index`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the length of this `ValueTuple` instance, which is always 0. There are no elements in a `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

0, the number of elements in this `ValueTuple` instance.

Implements

Length

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ValueTuple<T1> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with a single component.

C#

```
public struct ValueTuple<T1> : IComparable, IComparable<(T1)>,
IEquatable<(T1)>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's only element.

Inheritance Object → ValueType → ValueTuple<T1>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1>> , IEquatable<ValueTuple<T1>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1>(T1)	Initializes a new ValueTuple<T1> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1> instance's first element.

Methods

Expand table

CompareTo(ValueTuple<T1>)	Compares the current ValueTuple<T1> instance to a specified ValueTuple<T1> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object.
Equals(ValueTuple<T1>)	Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified ValueTuple<T1> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the `ValueTuple` element.

ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1>(ValueTuple<T1>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>(T1) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1> instance.

C#

```
public ValueTuple (T1 item1);
```

Parameters

`item1` T1

The value tuple's first element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1> instance's first element.

```
C#
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.CompareTo(ValueTuple<T1>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1> instance to a specified ValueTuple<T1> instance.

C#

```
public int CompareTo ((T1) other);
```

Parameters

other ValueTuple<T1>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object.
Equals(ValueTuple<T1>)	Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified ValueTuple<T1> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1> value type.
- It component is of the same types as that of the current instance.
- Its component is equal to that of the current instance. Equality is determined by the default equality comparer for the component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1>)

Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified ValueTuple<T1> instance.

C#

```
public bool Equals ((T1) other);
```

Parameters

`other` ValueTuple<T1>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1> instance.

Remarks

The string returned by this method takes the form (*Item1*, where *Item1* represents the value of the Item1 field. If the field value is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ValueTuple<T1>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the `ValueTuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the `ValueTuple` element. `index` must be 0.

Property Value

Object

The value of the `ValueTuple` element.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 0.

Remarks

The ValueTuple<T1>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1> object to a ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

C#

```
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

1, the number of elements in a ValueTuple<T1> object.

Implements

Length

Remarks

The ValueTuple<T1>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 2 components.

C#

```
public struct ValueTuple<T1,T2> : IComparable, IComparable<(T1, T2)>,
IEquatable<(T1, T2)>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

Inheritance Object → ValueType → ValueTuple<T1,T2>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2>> , IEquatable<ValueTuple<T1,T2>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2>(T1, T2)	Initializes a new ValueTuple<T1,T2> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2> instance's second element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2>)	Compares the current ValueTuple<T1,T2> instance to a specified ValueTuple<T1,T2> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object.
Equals(ValueTuple<T1,T2>)	Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified ValueTuple<T1,T2> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEquality	Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object based on a specified

Comparer)	comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2>(ValueTuple<T1,T2>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>(T1, T2) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2> instance.

C#

```
public ValueTuple (T1 item1, T2 item2);
```

Parameters

`item1` T1

The value tuple's first element.

`item2` T2

The value tuple's second element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ValueTuple<T1,T2>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2> instance's first element.

```
C#
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2> instance's second element.

```
C#
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.CompareTo(ValueTuple<T1,T2>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2> instance to a specified ValueTuple<T1,T2> instance.

C#

```
public int CompareTo ((T1, T2) other);
```

Parameters

`other` ValueTuple<T1,T2>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object.
Equals(ValueTuple<T1,T2>)	Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified ValueTuple<T1,T2> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2>)

Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified ValueTuple<T1,T2> instance.

C#

```
public bool Equals ((T1, T2) other);
```

Parameters

`other` ValueTuple<T1,T2>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, where *Item1* and *Item2* represent the values of the Item1 and Item2 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Remarks

This member is an explicit interface member implementation. It can be used only when the ValueTuple<T1,T2> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals(Object, Object) implementation is called only if `other` is not `null`, and if it can be successfully cast to a ValueTuple<T1,T2> whose components are of the same types as those of the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the ValueTuple<T1,T2> objects to be compared to the Equals(Object, Object) implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two ValueTuple<T1,T2> instances.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

```
C#
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 to 1.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 1.

Remarks

The ValueTuple<T1,T2>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2> object to an ITuple interface

object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

2, the number of elements in a ValueTuple<T1,T2> object.

Implements

Length

Remarks

The ValueTuple<T1,T2>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 3 components.

C#

```
public struct ValueTuple<T1,T2,T3> : IComparable, IComparable<(T1, T2, T3)>,
IEquatable<(T1, T2, T3)>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2,T3>> , IEquatable<ValueTuple<T1,T2,T3>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3>(T1, T2, T3)	Initializes a new ValueTuple<T1,T2,T3> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3> instance's third element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3>)	Compares the current ValueTuple<T1,T2,T3> instance to a specified ValueTuple<T1,T2,T3> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified ValueTuple<T1,T2,T3> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3> instance to a specified object by using a specified comparer and returns an integer that

	indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2,T3> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2,T3>(ValueTuple<T1,T2,T3>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ValueTuple<T1,T2,T3>(T1, T2, T3) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3> instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3> instance's first element.

C#

```
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3> instance's second element.

C#

```
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3> instance's third element.

C#

```
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.CompareTo(ValueTuple<T1,T2,T3>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3> instance to a specified ValueTuple<T1,T2,T3> instance.

C#

```
public int CompareTo ((T1, T2, T3) other);
```

Parameters

`other` ValueTuple<T1,T2,T3>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified ValueTuple<T1,T2,T3> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified ValueTuple<T1,T2,T3> instance.

C#

```
public bool Equals ((T1, T2, T3) other);
```

Parameters

`other` ValueTuple<T1,T2,T3>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, where *Item1*, *Item2*, and *Item3* represent the values of the Item1, Item2, and Item3 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

`other` Object

The object to compare with the current instance.

`comparer` IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 to 2.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 2.

Remarks

The ValueTuple<T1,T2,T3>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3> object to an ITuple

interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

3, the number of elements in a ValueTuple<T1,T2,T3> object.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 4 components.

C#

```
public struct ValueTuple<T1,T2,T3,T4> : IComparable, IComparable<(T1, T2,
T3, T4)>, IEquatable<(T1, T2, T3, T4)>,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

T4

The type of the value tuple's fourth element.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3,T4>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2,T3,T4>> , IEquatable<ValueTuple<T1,T2,T3,T4>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3,T4>(T1, T2, T3, T4)	Initializes a new ValueTuple<T1,T2,T3,T4> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's third element.
Item4	Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's fourth element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3,T4>)	Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified ValueTuple<T1,T2,T3,T4> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified ValueTuple<T1,T2,T3,T4> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2,T3,T4>(ValueTuple<T1,T2,T3,T4>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4>(T1, T2, T3, T4) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3,T4> instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3, T4 item4);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

item4 T4

The value tuple's fourth element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's first element.

C#

```
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's second element.

C#

```
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's third element.

C#

```
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.Item4 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4> instance's fourth element.

C#

```
public T4 Item4;
```

Field Value

T4

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.CompareTo(ValueTuple<T1,T2,T3,T4>) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified ValueTuple<T1,T2,T3,T4> instance.

C#

```
public int CompareTo ((T1, T2, T3, T4) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified ValueTuple<T1,T2,T3,T4> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3,T4> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3,T4>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified ValueTuple<T1,T2,T3,T4> instance.

C#

```
public bool Equals ((T1, T2, T3, T4) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3,T4> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3,T4> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, where *Item1*, *Item2*, *Item3*, and *Item4* represent the values of the Item1, Item2, Item3, and Item4 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

The object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3,T4> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 to 3.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 3.

Remarks

The ValueTuple<T1,T2,T3,T4>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4> object

to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

4, the number of elements in a ValueTuple<T1,T2,T3,T4> object.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3,T4>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 5 components.

C#

```
public struct ValueTuple<T1,T2,T3,T4,T5> : IComparable, IComparable<(T1, T2,
T3, T4, T5)>, IEquatable<(T1, T2, T3, T4, T5)>,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

T4

The type of the value tuple's fourth element.

T5

The type of the value tuple's fifth element.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3,T4,T5>

Implements IStructuralComparable , IStructuralEquatable , IComparable ,
IComparable<ValueTuple<T1,T2,T3,T4,T5>> ,

IEquatable<ValueTuple<T1,T2,T3,T4,T5>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5)	Initializes a new ValueTuple<T1,T2,T3,T4,T5> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's third element.
Item4	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's fourth element.
Item5	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's fifth element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3,T4,T5>)	Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified ValueTuple<T1,T2,T3,T4,T5> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5> instance.

GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2,T3,T4,T5>(ValueTuple<T1,T2,T3,T4,T5>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>(T1, T2, T3, T4, T5) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3,T4,T5> instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

item4 T4

The value tuple's fourth element.

item5 T5

The value tuple's fifth element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's first element.

C#

```
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's second element.

C#

```
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's third element.

C#

```
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Item4 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's fourth element.

C#

```
public T4 Item4;
```

Field Value

T4

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Item5 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5> instance's fifth element.

C#

```
public T5 Item5;
```

Field Value

T5

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified ValueTuple<T1,T2,T3,T4,T5> instance.

C#

```
public int CompareTo ((T1, T2, T3, T4, T5) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4,T5>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3,T4,T5> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3,T4,T5>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5> instance.

C#

```
public bool Equals ((T1, T2, T3, T4, T5) other);
```

Parameters

other ValueTuple<T1,T2,T3,T4,T5>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3,T4,T5> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4,T5>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3,T4,T5> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, *Item5*, where *Item1*, *Item2*, *Item3*, *Item4*, and *Item5* represent the values of the Item1, Item2, Item3, Item4, and Item5 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

The object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.

Value	Description
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3,T4,T5> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 to 4.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 4.

Remarks

The ValueTuple<T1,T2,T3,T4,T5>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5>

object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

5, the number of elements in a ValueTuple<T1,T2,T3,T4,T5> object.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3,T4,T5>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 6 components.

C#

```
public struct ValueTuple<T1,T2,T3,T4,T5,T6> : IComparable, IComparable<(T1,
T2, T3, T4, T5, T6)>, IEquatable<(T1, T2, T3, T4, T5, T6)>,
System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

T4

The type of the value tuple's fourth element.

T5

The type of the value tuple's fifth element.

T6

The type of the value tuple's sixth element.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3,T4,T5,T6>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2,T3,T4,T5,T6>> , IEquatable<ValueTuple<T1,T2,T3,T4,T5,T6>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6)	Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's third element.
Item4	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's fourth element.
Item5	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's fifth element.
Item6	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's sixth element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3,T4,T5,T6>)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object.

Equals(ValueTuple<T1,T2,T3,T4,T5,T6>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2,T3,T4,T5,T6>(ValueTuple<T1,T2,T3,T4,T5,T6>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>(T1, T2, T3, T4, T5, T6) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6
item6);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

item4 T4

The value tuple's fourth element.

item5 T5

The value tuple's fifth element.

item6 T6

The value tuple's sixth element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's first element.

C#

```
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's second element.

C#

```
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's third element.

C#

```
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item4 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's fourth element.

C#

```
public T4 Item4;
```

Field Value

T4

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item5 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's fifth element.

C#

```
public T5 Item5;
```

Field Value

T5

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Item6 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance's sixth element.

C#

```
public T6 Item6;
```

Field Value

T6

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public int CompareTo ((T1, T2, T3, T4, T5, T6) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4,T5,T6>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5,T6>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3,T4,T5,T6> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3,T4,T5,T6>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public bool Equals ((T1, T2, T3, T4, T5, T6) other);
```

Parameters

other ValueTuple<T1,T2,T3,T4,T5,T6>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4,T5,T6>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, and *Item6* represent the values of the Item1, Item2, Item3, Item4, Item5, and Item6 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6>.IStructural Comparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

The object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.

Value	Description
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6>.IStructural Equatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3,T4,T5,T6> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 to 5.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 5.

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6>

object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

```
C#
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

6, the number of elements in a ValueTuple<T1,T2,T3,T4,T5,T6> object.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a value tuple with 7 components.

C#

```
public struct ValueTuple<T1,T2,T3,T4,T5,T6,T7> : IComparable,
IComparable<(T1, T2, T3, T4, T5, T6, T7)>, IEquatable<(T1, T2, T3, T4, T5,
T6, T7)>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

T4

The type of the value tuple's fourth element.

T5

The type of the value tuple's fifth element.

T6

The type of the value tuple's sixth element.

T7

The type of the value tuple's seventh element.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3,T4,T5,T6,T7>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2,T3,T4,T5,T6,T7>> , IEquatable<ValueTuple<T1,T2,T3,T4,T5,T6,T7>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7)	Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's third element.
Item4	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's fourth element.
Item5	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's fifth element.
Item6	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's sixth element.
Item7	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's seventh element.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object, IComparer)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Extension Methods

Expand table

ToTuple<T1,T2,T3,T4,T5,T6,T7>(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Converts an instance of the `ValueTuple` structure to an instance of the `Tuple` class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6,T7>(T1, T2, T3, T4, T5, T6, T7) instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6
item6, T7 item7);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

item4 T4

The value tuple's fourth element.

item5 T5

The value tuple's fifth element.

item6 T6

The value tuple's sixth element.

item7 T7

The value tuple's seventh element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's first element.

C#

```
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's second element.

```
C#
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's third element.

C#

```
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item4 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's fourth element.

C#

```
public T4 Item4;
```

Field Value

T4

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item5 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's fifth element.

C#

```
public T5 Item5;
```

Field Value

T5

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item6 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's sixth element.

C#

```
public T6 Item6;
```

Field Value

T6

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.Item7 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance's seventh element.

C#

```
public T7 Item7;
```

Field Value

T7

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

C#

```
public int CompareTo ((T1, T2, T3, T4, T5, T6, T7) other);
```

Parameters

other ValueTuple<T1,T2,T3,T4,T5,T6,T7>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3,T4,T5,T6,T7> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

C#

```
public bool Equals ((T1, T2, T3, T4, T5, T6, T7) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4,T5,T6,T7>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, *Item7*, where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, and *Item7* represent the values of the Item1, Item2, Item3, Item4, Item5, Item6, and Item7 fields. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructural Comparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

The object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.

Value	Description
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructural Equatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

other Object

The object to compare with this instance.

comparer IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.IStructural Equatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.IComparable.CompareTo(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

The object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#
object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The index of the specified `ValueTuple` element. `index` can range from 0 (the index of Item1) to 6 (the index of Item7).

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0 or greater than 6.

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6,T7> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

C#

```
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

7, the number of elements in a ValueTuple<T1,T2,T3,T4,T5,T6,T7> object.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6,T7>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6,T7> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents an *n*-value tuple, where *n* is 8 or greater.

C#

```
public struct ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> : IComparable,
IComparable<(T1, T2, T3, T4, T5, T6, T7, TRest)>, IEquatable<(T1, T2, T3,
T4, T5, T6, T7, TRest)>, System.Collections.IStructuralComparable,
System.Collections.IStructuralEquatable,
System.Runtime.CompilerServices.ITuple where TRest : struct
```

Type Parameters

T1

The type of the value tuple's first element.

T2

The type of the value tuple's second element.

T3

The type of the value tuple's third element.

T4

The type of the value tuple's fourth element.

T5

The type of the value tuple's fifth element.

T6

The type of the value tuple's sixth element.

`T7`

The type of the value tuple's seventh element.

`TRest`

Any generic value tuple instance that defines the types of the tuple's remaining elements.

Inheritance Object → ValueType → ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>

Implements IStructuralComparable , IStructuralEquatable , IComparable , IComparable<ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>> , IEquatable<ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>> , ITuple

Remarks

For more information about this API, see Value tuples.

Constructors

Expand table

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>(T1, T2, T3, T4, T5, T6, T7, TRest)	Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Fields

Expand table

Item1	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's first element.
Item2	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's second element.
Item3	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's third element.
Item4	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's fourth element.

Item5	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's fifth element.
Item6	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's sixth element.
Item7	Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's seventh element.
Rest	Gets the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's remaining elements.

Methods

Expand table

CompareTo(ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.
Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.
GetHashCode()	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.
ToString()	Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Explicit Interface Implementations

Expand table

IComparable.CompareTo(Object)	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralComparable.CompareTo(Object,	Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance to a specified object by using a specified comparer and

IComparer)	returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.
IStructuralEquatable.Equals(Object, IEqualityComparer)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object based on a specified comparison method.
IStructuralEquatable.GetHashCode(IEqualityComparer)	Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance by using a specified computation method.
ITuple.Item[Int32]	Gets the value of the specified `ValueTuple` element.
ITuple.Length	Gets the number of elements in the `ValueTuple`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- ValueTuple

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

C#

```
public ValueTuple (T1 item1, T2 item2, T3 item3, T4 item4, T5 item5, T6
item6, T7 item7, TRest rest);
```

Parameters

item1 T1

The value tuple's first element.

item2 T2

The value tuple's second element.

item3 T3

The value tuple's third element.

item4 T4

The value tuple's fourth element.

item5 T5

The value tuple's fifth element.

item6 T6

The value tuple's sixth element.

item7 T7

The value tuple's seventh element.

`rest` TRest

An instance of any value tuple type that contains the values of the value's tuple's remaining elements.

Exceptions

ArgumentException

`rest` is not a generic value tuple type.

Remarks

In addition to using your language's build-in support for tuples, you can also create a value tuple by calling the static ValueTuple.Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8) method without having to explicitly specify the types of its elements.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Create<T1,T2,T3,T4,T5,T6,T7,T8>(T1, T2, T3, T4, T5, T6, T7, T8)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item1 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's first element.

```
C#
public T1 Item1;
```

Field Value

T1

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item2 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's second element.

```
C#
public T2 Item2;
```

Field Value

T2

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item3 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's third element.

```
C#
public T3 Item3;
```

Field Value

T3

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item4 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's fourth element.

```
C#
public T4 Item4;
```

Field Value

T4

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item5 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's fifth element.

```
C#
public T5 Item5;
```

Field Value

T5

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item6 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's sixth element.

C#

```
public T6 Item6;
```

Field Value

T6

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Item7 Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's seventh element.

```
C#
public T7 Item7;
```

Field Value

T7

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Rest Field

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's remaining elements.

```
C#
public TRest Rest;
```

Field Value

TRest

Remarks

The Rest field returns a nested value tuple that allows access to the eighth though *n*th elements of the tuple. Depending on the total number of elements in the tuple, the values of the eighth through fourteenth components can be retrieved from the nested value tuple's `Item1` through `Item7` properties. You can then use the Rest property of a nested ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to retrieve the value tuple at the next level of nesting.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

C#

```
public int CompareTo ((T1, T2, T3, T4, T5, T6, T7, TRest) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>

The tuple to compare with this instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(T)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object.

Overloads

Expand table

Equals(Object)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object.
Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>)	Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Equals(Object)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

`obj` Object

The object to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified object; otherwise, `false`.

Remarks

The `obj` argument is considered to be equal to the current instance under the following conditions:

- It is a ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> value type.
- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Equals(ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>)

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

```
C#
```

```
public bool Equals ((T1, T2, T3, T4, T5, T6, T7, TRest) other);
```

Parameters

`other` ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>

The value tuple to compare with this instance.

Returns

Boolean

`true` if the current instance is equal to the specified tuple; otherwise, `false`.

Implements

Equals(T)

Remarks

The `other` argument is considered to be equal to the current instance under the following conditions:

- Its components are of the same types as those of the current instance.
- Its components are equal to those of the current instance. Equality is determined by the default equality comparer for each component.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

The hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.To String Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a string that represents the value of this ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

C#

```
public override string ToString ();
```

Returns

String

The string representation of this ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Remarks

The string returned by this method takes the form (*Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, *Item7*, *Item8*...), where *Item1*, *Item2*, *Item3*, *Item4*, *Item5*, *Item6*, and *Item7* represent the values of the Item1, Item2, Item3, Item4, Item5, Item6, and Item7 fields. *Item8* represents the value of the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance's `Next.Item1` field. The value of any additional nested elements follows *Item8*. If any of the field values is `null`, it is represented as String.Empty.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance to a specified object by using a specified comparer and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IStructuralComparable.CompareTo (object other,
System.Collections.IComparer comparer);
```

Parameters

other Object

The object to compare with the current instance.

comparer IComparer

An object that provides custom rules for comparison.

Returns

Int32

A signed integer that indicates the relative position of this instance and `other` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object, IComparer)

Exceptions

ArgumentException

`other` is not a ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Remarks

This member is an explicit interface implementation. It can only be used when the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralComparable interface.

This method lets you define customized comparisons of ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects. For example, you can use this method to order ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects based on the value of a specific component.

Although this method can be called directly, it is most commonly called by collection-sorting methods that include IComparer parameters to order the members of a collection. For example, it is called by the Array.Sort(Array, IComparer) method and the Add method of a SortedList object that is instantiated by using the SortedList.SortedList(IComparer) constructor.

> ⊗ **Caution**
>
> The **IStructuralComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **IStructuralEquatable.Equals** method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralEquatable.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value that indicates whether the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is equal to a specified object based on a specified comparison method.

C#

```
bool IStructuralEquatable.Equals (object other,
System.Collections.IEqualityComparer comparer);
```

Parameters

`other` Object

The object to compare with this instance.

`comparer` IEqualityComparer

An object that defines the method to use to evaluate whether the two objects are equal.

Returns

Boolean

`true` if the current instance is equal to the specified objects; otherwise, `false`.

Implements

Equals(Object, IEqualityComparer)

Remarks

This member is an explicit interface implementation. It can only be used when the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralEquatable interface.

The IEqualityComparer.Equals implementation is called only if `other` is not `null`, and if it can be successfully cast to a ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object that has the same total number of components (including those in nested tuple objects) of the same types as the current instance. The IStructuralEquatable.Equals(Object, IEqualityComparer) method first passes the Item1 values of the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects to be compared to the IEqualityComparer.Equals implementation. If this method call returns `true`, the method is called again and passed the Item2 values of the two ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects. This continues until the method call returns `false` when it compares a specific pair of values, or the two Rest values are passed to the method.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IStructuralEquatable.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Calculates the hash code for the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance by using a specified computation method.

C#

```
int IStructuralEquatable.GetHashCode (System.Collections.IEqualityComparer comparer);
```

Parameters

`comparer` IEqualityComparer

An object whose GetHashCode(Object) method calculates the hash code of the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance.

Returns

Int32

A 32-bit signed integer hash code.

Implements

GetHashCode(IEqualityComparer)

Remarks

This member is an explicit interface implementation. It can only be used when the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IStructuralEquatable

interface. The method simply wraps a call to the `comparer` object's IEqualityComparer.GetHashCode implementation.

The algorithm used to compute the hash code should return the same hash code for two ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> objects that are considered to be equal.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IComparable.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to a specified object and returns an integer that indicates whether the current object is before, after, or in the same position as the specified object in the sort order.

C#

```
int IComparable.CompareTo (object other);
```

Parameters

`other` Object

An object to compare with the current instance.

Returns

Int32

A signed integer that indicates the relative position of this instance and `obj` in the sort order, as shown in the following table.

Expand table

Value	Description
A negative integer	This instance precedes `other`.
Zero	This instance and `other` have the same position in the sort order.
A positive integer	This instance follows `other`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`other` is not a ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object.

Remarks

This member is an explicit interface member implementation. It can be used only when the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> instance is cast to an IComparable interface.

This method provides the IComparable.CompareTo implementation for the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> structure. Although the method can be called directly, it is most commonly called by the default overloads of collection-sorting methods, such as Array.Sort(Array) and SortedList.Add, to order the members of a collection.

> **⊗ Caution**
>
> The **ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.IComparable.CompareTo** method is intended for use in sorting operations. It should not be used when the primary purpose of a comparison is to determine whether two objects are equal. To determine whether two objects are equal, call the **ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.Equals(Object)** method.

This method uses the default object comparer to compare each component.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Item[Int32] Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the specified `ValueTuple` element.

```
C#

object? System.Runtime.CompilerServices.ITuple.Item[int index] { get; }
```

Parameters

`index` Int32

The value of the specified `ValueTuple` element. `index` can range from 0 for `Item1` to one less than the number of elements in the `ValueTuple`.

Property Value

Object

The value of the `ValueTuple` element at the specified position.

Implements

Item[Int32]

Exceptions

IndexOutOfRangeException

`index` is less than 0.

-or-

`index` is greater than or equal to ITuple.Length.

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Item[] property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Value Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Length Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the number of elements in the `ValueTuple`.

C#

```
int System.Runtime.CompilerServices.ITuple.Length { get; }
```

Property Value

Int32

The number of elements in the `ValueTuple`.

Implements

Length

Remarks

The ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>.ITuple.Length property is an explicit interface implementation. To call it, you must cast or convert the ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest> object to an ITuple interface object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueType Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Provides the base class for value types.

```
C#
public abstract class ValueType
```

Inheritance Object → ValueType

Derived _GUID
Accessibility.__MIDL_IWinTypes_0009
Accessibility._RemotableHandle
IStream
ISymUnmanagedBinder
More...

Remarks

ValueType overrides the virtual methods from Object with more appropriate implementations for value types. See also Enum, which inherits from ValueType.

Data types are separated into value types and reference types. Value types are either stack-allocated or allocated inline in a structure. Reference types are heap-allocated. Both reference and value types are derived from the ultimate base class Object. In cases where it is necessary for a value type to behave like an object, a wrapper that makes the value type look like a reference object is allocated on the heap, and the value type's value is copied into it. The wrapper is marked so the system knows that it contains a value type. This process is known as boxing, and the reverse process is known as unboxing. Boxing and unboxing allow any type to be treated as an object.

Although ValueType is the implicit base class for value types, you cannot create a class that inherits from ValueType directly. Instead, individual compilers provide a language

keyword or construct (such as `struct` in C# and `Structure`...`End Structure` in Visual Basic) to support the creation of value types.

Aside from serving as the base class for value types in the .NET Framework, the ValueType structure is generally not used directly in code. However, it can be used as a parameter in method calls to restrict possible arguments to value types instead of all objects, or to permit a method to handle a number of different value types. The following example illustrates how ValueType prevents reference types from being passed to methods. It defines a class named `Utility` that contains four methods: `IsNumeric`, which indicates whether its argument is a number; `IsInteger`, which indicates whether its argument is an integer; `IsFloat`, which indicates whether its argument is a floating-point number; and `Compare`, which indicates the relationship between two numeric values. In each case, the method parameters are of type ValueType, and reference types are prevented from being passed to the methods.

C#

```
using System;
using System.Numerics;

public class Utility
{
   public enum NumericRelationship {
      GreaterThan = 1,
      EqualTo = 0,
      LessThan = -1
   };

   public static NumericRelationship Compare(ValueType value1, ValueType
value2)
   {
      if (! IsNumeric(value1))
         throw new ArgumentException("value1 is not a number.");
      else if (! IsNumeric(value2))
         throw new ArgumentException("value2 is not a number.");

      // Use BigInteger as common integral type
      if (IsInteger(value1) && IsInteger(value2)) {
         BigInteger bigint1 = (BigInteger) value1;
         BigInteger bigint2 = (BigInteger) value2;
         return (NumericRelationship) BigInteger.Compare(bigint1, bigint2);
      }
      // At least one value is floating point; use Double.
      else {
         Double dbl1 = 0;
         Double dbl2 = 0;
         try {
            dbl1 = Convert.ToDouble(value1);
         }
```

```
            catch (OverflowException) {
               Console.WriteLine("value1 is outside the range of a Double.");
            }
            try {
               dbl2 = Convert.ToDouble(value2);
            }
            catch (OverflowException) {
               Console.WriteLine("value2 is outside the range of a Double.");
            }
            return (NumericRelationship) dbl1.CompareTo(dbl2);
         }
      }

      public static bool IsInteger(ValueType value)
      {
         return (value is SByte || value is Int16 || value is Int32
                 || value is Int64 || value is Byte || value is UInt16
                 || value is UInt32 || value is UInt64
                 || value is BigInteger);
      }

      public static bool IsFloat(ValueType value)
      {
         return (value is float | value is double | value is Decimal);
      }

      public static bool IsNumeric(ValueType value)
      {
         return (value is Byte ||
                 value is Int16 ||
                 value is Int32 ||
                 value is Int64 ||
                 value is SByte ||
                 value is UInt16 ||
                 value is UInt32 ||
                 value is UInt64 ||
                 value is BigInteger ||
                 value is Decimal ||
                 value is Double ||
                 value is Single);
      }
   }
```

The following example illustrates calls to the methods of the `Utility` class.

C#

```
public class Example
{
   public static void Main()
   {
      Console.WriteLine(Utility.IsNumeric(12));
      Console.WriteLine(Utility.IsNumeric(true));
```

```
        Console.WriteLine(Utility.IsNumeric('c'));
        Console.WriteLine(Utility.IsNumeric(new DateTime(2012, 1, 1)));
        Console.WriteLine(Utility.IsInteger(12.2));
        Console.WriteLine(Utility.IsInteger(123456789));
        Console.WriteLine(Utility.IsFloat(true));
        Console.WriteLine(Utility.IsFloat(12.2));
        Console.WriteLine(Utility.IsFloat(12));
        Console.WriteLine("{0} {1} {2}", 12.1, Utility.Compare(12.1, 12), 12);
    }
}
// The example displays the following output:
//       True
//       False
//       False
//       False
//       False
//       True
//       False
//       True
//       False
//       12.1 GreaterThan 12
```

Constructors

Expand table

ValueType()	Initializes a new instance of the ValueType class.

Methods

Expand table

Equals(Object)	Indicates whether this instance and a specified object are equal.
GetHashCode()	Returns the hash code for this instance.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns the fully qualified type name of this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Object
- Enum

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueType Constructor

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the ValueType class.

C#

```
protected ValueType ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueType.Equals(Object) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Indicates whether this instance and a specified object are equal.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

The object to compare with the current instance.

Returns

Boolean

`true` if `obj` and this instance are the same type and represent the same value; otherwise, `false`.

Examples

The following example demonstrates how the Equals method can be overridden by a derived value type.

C#

```
public struct Complex
{
    public double m_Re;
    public double m_Im;

    public override bool Equals( object ob ){
        if( ob is Complex ) {
            Complex c = (Complex) ob;
            return m_Re==c.m_Re && m_Im==c.m_Im;
```

```
        }
        else {
            return false;
        }
    }

    public override int GetHashCode(){
        return m_Re.GetHashCode() ^ m_Im.GetHashCode();
    }
}
```

Remarks

The ValueType.Equals(Object) method overrides Object.Equals(Object) and provides the default implementation of value equality for all value types in .NET.

The default implementation calls Object.Equals(Object) on each field of the current instance and `obj` and returns `true` if all fields are equal.

> **Tip**
>
> Particularly if your value type contains fields that are reference types, you should override the **Equals(Object)** method. This can improve performance and enable you to more closely represent the meaning of equality for the type.

Notes for the Windows Runtime

When you call the Equals method on a Windows Runtime structure, it provides the default behavior for value types that don't override Equals. This is part of the support that .NET provides for the Windows Runtime (see .NET Support for Windows Store Apps and Windows Runtime). Windows Runtime structures can't override Equals, even if they're written with C# or Visual Basic, because they can't have methods. (In addition, structures in the Windows Runtime itself don't inherit ValueType.) However, they appear to have ToString, Equals, and GetHashCode methods when you use them in your C# or Visual Basic code, and .NET provides the default behavior for these methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ValueType.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the hash code for this instance.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer that is the hash code for this instance.

Examples

The following example demonstrates how the GetHashCode method can be overridden by a derived value type.

C#

```
public struct Complex
{
    public double m_Re;
    public double m_Im;

    public override bool Equals( object ob ){
        if( ob is Complex ) {
            Complex c = (Complex) ob;
            return m_Re==c.m_Re && m_Im==c.m_Im;
        }
        else {
            return false;
        }
    }

    public override int GetHashCode(){
        return m_Re.GetHashCode() ^ m_Im.GetHashCode();
```

```
    }
}
```

Remarks

The GetHashCode method applies to types derived from ValueType. One or more fields of the derived type is used to calculate the return value. If you call the derived type's `GetHashCode` method, the return value is not likely to be suitable for use as a key in a hash table. Additionally, if the value of one or more of those fields changes, the return value might become unsuitable for use as a key in a hash table. In either case, consider writing your own implementation of the GetHashCode method that more closely represents the concept of a hash code for the type.

For more information, see Object.GetHashCode, and System.Collections.Hashtable.

Notes for the Windows Runtime

When you call the GetHashCode method on a Windows Runtime structure, it provides the default behavior for value types that don't override GetHashCode. This is part of the support that .NET provides for the Windows Runtime (see .NET Support for Windows Store Apps and Windows Runtime). Windows Runtime structures can't override GetHashCode, even if they're written with C# or Visual Basic, because they can't have methods. (In addition, structures in the Windows Runtime itself don't inherit ValueType.) However, they appear to have ToString, Equals, and GetHashCode methods when you use them in your C# or Visual Basic code, and .NET provides the default behavior for these methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ValueType.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns the fully qualified type name of this instance.

C#

```
public override string? ToString ();
```

Returns

String

The fully qualified type name.

Remarks

The ValueType.ToString method overrides the Object.ToString method and provides the default implementation of the `ToString` method for value types. (Value types are types defined by the `struct` keyword in C#, and by the `Structure`...`End Structure` construct in Visual Basic.) Functionally, however, the implementation is that same as that of Object.ToString: the method returns the fully qualified type name.

Value types defined by the `struct` keyword in C# and the `Structure`...`End Structure` construct in Visual Basic typically override the ValueType.ToString method to provide a more meaningful string representation of the value type. The following example illustrates the difference. It defines two value types, `EmployeeA` and `EmployeeB`, creates an instance of each, and calls its `ToString` method. Because the `EmployeeA` structure does not override the ValueType.ToString method, it displays only the fully qualified type name. The `EmployeeB.ToString` method, on the other hand, provides meaningful information about the object.

C#

```
using System;
using Corporate.EmployeeObjects;

public class Example
{
   public static void Main()
   {
      var empA = new EmployeeA{ Name = "Robert",};
      Console.WriteLine(empA.ToString());

      var empB = new EmployeeB{ Name = "Robert",};
      Console.WriteLine(empB.ToString());
   }
}

namespace Corporate.EmployeeObjects
{
    public struct EmployeeA
    {
         public String Name { get; set; }
    }

    public struct EmployeeB
    {
         public String Name { get; set; }

         public override String ToString()
         {
              return Name;
         }
    }
}
// The example displays the following output:
//     Corporate.EmployeeObjects.EmployeeA
//     Robert
```

Note that, although enumeration types are also value types, they derive from the Enum class, which overrides ValueType.ToString.

Notes for the Windows Runtime

When you call the ToString method on a Windows Runtime structure, it provides the default behavior for value types that don't override ToString. This is part of the support that .NET provides for the Windows Runtime (see .NET Support for Windows Store Apps and Windows Runtime). Windows Runtime structures can't override ToString, even if they're written with C# or Visual Basic, because they can't have methods. (In addition, structures in the Windows Runtime itself don't inherit ValueType.) However, they appear

to have ToString, Equals, and GetHashCode methods when you use them in your C# or Visual Basic code, and .NET provides the default behavior for these methods.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents the version number of an assembly, operating system, or the common language runtime. This class cannot be inherited.

```C#
public sealed class Version : ICloneable, IComparable, IComparable<Version>, IEquatable<Version>, ISpanFormattable, IUtf8SpanFormattable
```

Inheritance Object → Version

Implements IComparable , IComparable<Version> , IEquatable<Version> , ICloneable , IFormattable , ISpanFormattable , IUtf8SpanFormattable

Examples

The following example uses the AssemblyVersionAttribute attribute to assign a version number to an assembly. At compile time, this version information is stored with the assembly's metadata. At run time, the example retrieves the value of the Type.Assembly property on a type found in the assembly to get a reference to the executing assembly, and it retrieves the assembly's version information from the Version property of the AssemblyName object returned by the Assembly.GetName method.

```C#
using System;
using System.Reflection;

[assembly:AssemblyVersionAttribute("2.0.1")]

public class Example1
{
   public static void Main()
   {
      Assembly thisAssem = typeof(Example1).Assembly;
```

```
        AssemblyName thisAssemName = thisAssem.GetName();

        Version ver = thisAssemName.Version;

        Console.WriteLine("This is version {0} of {1}.", ver,
thisAssemName.Name);
    }
}
// The example displays the following output:
//         This is version 2.0.1.0 of Example1.
```

Remarks

For more information about this API, see Supplemental API remarks for Version.

Constructors

Expand table

Version()	Initializes a new instance of the Version class.
Version(Int32, Int32)	Initializes a new instance of the Version class using the specified major and minor values.
Version(Int32, Int32, Int32)	Initializes a new instance of the Version class using the specified major, minor, and build values.
Version(Int32, Int32, Int32, Int32)	Initializes a new instance of the Version class with the specified major, minor, build, and revision numbers.
Version(String)	Initializes a new instance of the Version class using the specified string.

Properties

Expand table

Build	Gets the value of the build component of the version number for the current Version object.
Major	Gets the value of the major component of the version number for the current Version object.
Major Revision	Gets the high 16 bits of the revision number.

Minor	Gets the value of the minor component of the version number for the current Version object.
Minor Revision	Gets the low 16 bits of the revision number.
Revision	Gets the value of the revision component of the version number for the current Version object.

Methods

Expand table

Clone()	Returns a new Version object whose value is the same as the current Version object.
CompareTo(Object)	Compares the current Version object to a specified object and returns an indication of their relative values.
CompareTo(Version)	Compares the current Version object to a specified Version object and returns an indication of their relative values.
Equals(Object)	Returns a value indicating whether the current Version object is equal to a specified object.
Equals(Version)	Returns a value indicating whether the current Version object and a specified Version object represent the same value.
GetHashCode()	Returns a hash code for the current Version object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Parse(ReadOnlySpan<Char>)	Converts the specified read-only span of characters that represents a version number to an equivalent Version object.
Parse(String)	Converts the string representation of a version number to an equivalent Version object.
ToString()	Converts the value of the current Version object to its equivalent String representation.
ToString(Int32)	Converts the value of the current Version object to its equivalent String representation. A specified count indicates the number of components to return.

TryFormat(Span<Byte>, Int32)	Tries to format this version instance into a span of bytes.
TryFormat(Span<Byte>, Int32, Int32)	Tries to format this version instance into a span of bytes.
TryFormat(Span<Char>, Int32)	Tries to format this version instance into a span of characters.
TryFormat(Span<Char>, Int32, Int32)	Tries to format this version instance into a span of characters.
TryParse(ReadOnlySpan<Char>, Version)	Tries to convert the specified read-only span of characters representing a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.
TryParse(String, Version)	Tries to convert the string representation of a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.

Operators

Expand table

Equality(Version, Version)	Determines whether two specified Version objects are equal.
GreaterThan(Version, Version)	Determines whether the first specified Version object is greater than the second specified Version object.
GreaterThanOrEqual(Version, Version)	Determines whether the first specified Version object is greater than or equal to the second specified Version object.
Inequality(Version, Version)	Determines whether two specified Version objects are not equal.
LessThan(Version, Version)	Determines whether the first specified Version object is less than the second specified Version object.
LessThanOrEqual(Version, Version)	Determines whether the first specified Version object is less than or equal to the second Version object.

Explicit Interface Implementations

Expand table

IFormattable.ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.

ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.
IUtf8SpanFormattable.TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the Version class with the specified major, minor, build, and revision numbers.

Overloads

Expand table

Version()	Initializes a new instance of the Version class.
Version(String)	Initializes a new instance of the Version class using the specified string.
Version(Int32, Int32)	Initializes a new instance of the Version class using the specified major and minor values.
Version(Int32, Int32, Int32)	Initializes a new instance of the Version class using the specified major, minor, and build values.
Version(Int32, Int32, Int32, Int32)	Initializes a new instance of the Version class with the specified major, minor, build, and revision numbers.

Version()

Initializes a new instance of the Version class.

C#

```
public Version ();
```

Remarks

This constructor creates a Version object with the following property values.

Expand table

Property	Value
Major	0
Minor	0
Build	undefined (-1)
Revision	undefined (-1)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Version(String)

Initializes a new instance of the Version class using the specified string.

C#

```
public Version (string version);
```

Parameters

`version` String

A string containing the major, minor, build, and revision numbers, where each number is delimited with a period character ('.').

Exceptions

ArgumentException

`version` has fewer than two components or more than four components.

ArgumentNullException

`version` is `null`.

ArgumentOutOfRangeException

A major, minor, build, or revision component is less than zero.

FormatException

At least one component of `version` does not parse to an integer.

OverflowException

At least one component of `version` represents a number greater than Int32.MaxValue.

Remarks

The `version` parameter can contain only the components major, minor, build, and revision, in that order, and all separated by periods. There must be at least two components, and at most four. The first two components are assumed to be major and minor. The value of unspecified components is undefined.

The format of the version number is as follows. Optional components are shown in square brackets ('[' and ']'):

major.minor[.build[.revision]]

All defined components must be integers greater than or equal to 0. For example, if the major number is 6, the minor number is 2, the build number is 1, and the revision number is 3, then `version` should be "6.2.1.3".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Version(Int32, Int32)

Initializes a new instance of the Version class using the specified major and minor values.

```
C#
public Version (int major, int minor);
```

Parameters

`major` Int32

The major version number.

`minor` Int32

The minor version number.

Exceptions

ArgumentOutOfRangeException

`major` or `minor` is less than zero.

Remarks

This constructor creates a Version object with the following property values.

Expand table

Property	Value
Major	`major`
Minor	`minor`
Build	undefined (-1)
Revision	undefined (-1)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Version(Int32, Int32, Int32)

Initializes a new instance of the Version class using the specified major, minor, and build values.

C#

```
public Version (int major, int minor, int build);
```

Parameters

`major` Int32

The major version number.

`minor` Int32

The minor version number.

`build` Int32

The build number.

Exceptions

ArgumentOutOfRangeException

`major`, `minor`, or `build` is less than zero.

Remarks

This constructor creates a Version object with the following property values.

Expand table

Property	Value
Major	`major`
Minor	`minor`
Build	`build`
Revision	undefined (-1)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Version(Int32, Int32, Int32, Int32)

Initializes a new instance of the Version class with the specified major, minor, build, and revision numbers.

C#

```
public Version (int major, int minor, int build, int revision);
```

Parameters

`major` Int32

The major version number.

`minor` Int32

The minor version number.

`build` Int32

The build number.

`revision` Int32

The revision number.

Exceptions

ArgumentOutOfRangeException

`major`, `minor`, `build`, or `revision` is less than zero.

Examples

The following code example demonstrates the Version constructor, and Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);

    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor,
interim.Build,
                              interim.MajorRevision,
interim.MinorRevision);
    }
}
/*
```

```
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

This constructor creates a Version object with the following property values.

Expand table

Property	Value
Major	`major`
Minor	`minor`
Build	`build`
Revision	`revision`

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.Build Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the build component of the version number for the current Version object.

C#

```
public int Build { get; }
```

Property Value

Int32

The build number, or -1 if the build number is undefined.

Examples

The following code example demonstrates the Version constructor, and Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
```

```
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);

    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

For example, if the version number is 6.2.1.3, the build number is 1. If the version number is 6.2, the build number is undefined.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.Major Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the major component of the version number for the current Version object.

C#

```
public int Major { get; }
```

Property Value

Int32

The major version number.

Remarks

For example, if the version number is 6.2, the major version is 6.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.MajorRevision Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the high 16 bits of the revision number.

C#

```
public short MajorRevision { get; }
```

Property Value

Int16

A 16-bit signed integer.

Examples

The following code example demonstrates the Version constructor, and the Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);
```

```
    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

Suppose you release an interim version of your application to temporarily correct a problem until you can release a permanent solution. The temporary version does not warrant a new revision number, but it does need to be identified as a different version. In this case, encode the identification information in the high and low 16-bit portions of the 32-bit revision number. Use the Revision property to obtain the entire revision number, use the MajorRevision property to obtain the high 16 bits, and use the MinorRevision property to obtain the low 16 bits.

Starting in the .NET Framework version 2.0, the Windows NT operating system uses the MajorRevision property to encode the service pack number.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MinorRevision

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.Minor Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the minor component of the version number for the current Version object.

C#

```
public int Minor { get; }
```

Property Value

Int32

The minor version number.

Examples

The following code example demonstrates the Version constructor, and Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
```

```
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);

    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

For example, if the version number is 6.2, the minor version is 2.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.MinorRevision Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the low 16 bits of the revision number.

C#

```
public short MinorRevision { get; }
```

Property Value

Int16

A 16-bit signed integer.

Examples

The following code example demonstrates the Version constructor, and the Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);
```

```
    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

Suppose you release an interim version of your application to temporarily correct a problem until you can release a permanent solution. The temporary version does not warrant a new revision number, but it does need to be identified as a different version. In this case, encode the identification information in the high and low 16-bit portions of the 32-bit revision number. Use the Revision property to obtain the entire revision number, use the MajorRevision property to obtain the high 16 bits, and use the MinorRevision property to obtain the low 16 bits.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- MajorRevision

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.Revision Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets the value of the revision component of the version number for the current Version object.

C#

```
public int Revision { get; }
```

Property Value

Int32

The revision number, or -1 if the revision number is undefined.

Examples

The following code example demonstrates the Version constructor, and Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
```

```
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);

    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Remarks

For example, if the version number is 6.2.1.3, the revision number is 3. If the version number is 6.2, the revision number is undefined.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.Clone Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a new Version object whose value is the same as the current Version object.

C#

```
public object Clone ();
```

Returns

Object

A new Object whose values are a copy of the current Version object.

Implements

Clone()

Remarks

If the type of the return value is important, cast the Object instance that is returned by this method to a Version object.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.CompareTo Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Compares the current Version object to a specified object or Version and returns an indication of their relative values.

Overloads

Expand table

CompareTo(Version)	Compares the current Version object to a specified Version object and returns an indication of their relative values.
CompareTo(Object)	Compares the current Version object to a specified object and returns an indication of their relative values.

CompareTo(Version)

Compares the current Version object to a specified Version object and returns an indication of their relative values.

```
C#

public int CompareTo (Version? value);
```

Parameters

`value` Version

A Version object to compare to the current Version object, or `null`.

Returns

Int32

A signed integer that indicates the relative values of the two objects, as shown in the following table.

Expand table

Return value	Meaning
Less than zero	The current Version object is a version before `value`.
Zero	The current Version object is the same version as `value`.
Greater than zero	The current Version object is a version subsequent to `value`, or `value` is `null`.

Implements

CompareTo(T)

Examples

The following code example demonstrates generic and nongeneric versions of the `CompareTo` method for several value and reference types.

C#

```
// This example demonstrates the generic and non-generic versions of the
// CompareTo method for several base types.
// The non-generic version takes a parameter of type Object, while the
generic
// version takes a type-specific parameter, such as Boolean, Int32, or
Double.

using System;

class Sample
{
    public static void Main()
    {
    string    nl = Environment.NewLine;
    string    msg = "{0}The following is the result of using the generic
and non-generic{0}" +
                    "versions of the CompareTo method for several base
types:{0}";

    DateTime  now = DateTime.Now;
// Time span = 11 days, 22 hours, 33 minutes, 44 seconds
    TimeSpan  tsX = new TimeSpan(11, 22, 33, 44);
// Version = 1.2.333.4
    Version   versX = new Version("1.2.333.4");
```

```
// Guid = CA761232-ED42-11CE-BACD-00AA0057B223
    Guid      guidX = new Guid("{CA761232-ED42-11CE-BACD-00AA0057B223}");

    Boolean  a1 = true,  a2 = true;
    Byte     b1 = 1,     b2 = 1;
    Int16    c1 = -2,    c2 = 2;
    Int32    d1 = 3,     d2 = 3;
    Int64    e1 = 4,     e2 = -4;
    Decimal  f1 = -5.5m, f2 = 5.5m;
    Single   g1 = 6.6f,  g2 = 6.6f;
    Double   h1 = 7.7d,  h2 = -7.7d;
    Char     i1 = 'A',   i2 = 'A';
    String   j1 = "abc", j2 = "abc";
    DateTime k1 = now,   k2 = now;
    TimeSpan l1 = tsX,   l2 = tsX;
    Version  m1 = versX, m2 = new Version("2.0");
    Guid     n1 = guidX, n2 = guidX;

// The following types are not CLS-compliant.
    SByte    w1 = 8,     w2 = 8;
    UInt16   x1 = 9,     x2 = 9;
    UInt32   y1 = 10,    y2 = 10;
    UInt64   z1 = 11,    z2 = 11;
//
    Console.WriteLine(msg, nl);
    try
        {
// The second and third Show method call parameters are automatically
boxed because
// the second and third Show method declaration arguments expect type
Object.

        Show("Boolean:  ", a1, a2, a1.CompareTo(a2),
a1.CompareTo((Object)a2));
        Show("Byte:     ", b1, b2, b1.CompareTo(b2),
b1.CompareTo((Object)b2));
        Show("Int16:    ", c1, c2, c1.CompareTo(c2),
c1.CompareTo((Object)c2));
        Show("Int32:    ", d1, d2, d1.CompareTo(d2),
d1.CompareTo((Object)d2));
        Show("Int64:    ", e1, e2, e1.CompareTo(e2),
e1.CompareTo((Object)e2));
        Show("Decimal:  ", f1, f2, f1.CompareTo(f2),
f1.CompareTo((Object)f2));
        Show("Single:   ", g1, g2, g1.CompareTo(g2),
g1.CompareTo((Object)g2));
        Show("Double:   ", h1, h2, h1.CompareTo(h2),
h1.CompareTo((Object)h2));
        Show("Char:     ", i1, i2, i1.CompareTo(i2),
i1.CompareTo((Object)i2));
        Show("String:   ", j1, j2, j1.CompareTo(j2),
j1.CompareTo((Object)j2));
        Show("DateTime: ", k1, k2, k1.CompareTo(k2),
k1.CompareTo((Object)k2));
        Show("TimeSpan: ", l1, l2, l1.CompareTo(l2),
```

```
l1.CompareTo((Object)l2));
        Show("Version:  ", m1, m2, m1.CompareTo(m2),
m1.CompareTo((Object)m2));
        Show("Guid:     ", n1, n2, n1.CompareTo(n2),
n1.CompareTo((Object)n2));
//
        Console.WriteLine("{0}The following types are not CLS-
compliant:", nl);
        Show("SByte:    ", w1, w2, w1.CompareTo(w2),
w1.CompareTo((Object)w2));
        Show("UInt16:   ", x1, x2, x1.CompareTo(x2),
x1.CompareTo((Object)x2));
        Show("UInt32:   ", y1, y2, y1.CompareTo(y2),
y1.CompareTo((Object)y2));
        Show("UInt64:   ", z1, z2, z1.CompareTo(z2),
z1.CompareTo((Object)z2));
        }
    catch (Exception e)
        {
        Console.WriteLine(e);
        }
    }

    public static void Show(string caption, Object var1, Object var2,
                            int resultGeneric, int resultNonGeneric)
    {
    string relation;

    Console.Write(caption);
    if (resultGeneric == resultNonGeneric)
        {
        if      (resultGeneric < 0) relation = "less than";
        else if (resultGeneric > 0) relation = "greater than";
        else                        relation = "equal to";
        Console.WriteLine("{0} is {1} {2}", var1, relation, var2);
        }

// The following condition will never occur because the generic and non-
generic
// CompareTo methods are equivalent.

    else
        {
        Console.WriteLine("Generic CompareTo = {0}; non-generic CompareTo
= {1}",
                          resultGeneric, resultNonGeneric);
        }
   }
}
/*
This example produces the following results:

The following is the result of using the generic and non-generic versions
of the
CompareTo method for several base types:
```

```
Boolean:  True is equal to True
Byte:     1 is equal to 1
Int16:    -2 is less than 2
Int32:    3 is equal to 3
Int64:    4 is greater than -4
Decimal:  -5.5 is less than 5.5
Single:   6.6 is equal to 6.6
Double:   7.7 is greater than -7.7
Char:     A is equal to A
String:   abc is equal to abc
DateTime: 12/1/2003 5:37:46 PM is equal to 12/1/2003 5:37:46 PM
TimeSpan: 11.22:33:44 is equal to 11.22:33:44
Version:  1.2.333.4 is less than 2.0
Guid:     ca761232-ed42-11ce-bacd-00aa0057b223 is equal to ca761232-ed42-
11ce-bacd-00
aa0057b223

The following types are not CLS-compliant:
SByte:    8 is equal to 8
UInt16:   9 is equal to 9
UInt32:   10 is equal to 10
UInt64:   11 is equal to 11
*/
```

Remarks

This method implements the System.IComparable<T> interface, and performs slightly better than the CompareTo method because it does not have to unbox the `value` parameter.

The components of `Version` in decreasing order of importance are: major, minor, build, and revision. An unknown component is assumed to be older than any known component. For example:

- Version 1.1 is older than version 1.1.0.
- Version 1.1 is older than version 1.1.1.
- Version 1.1 is older than version 1.1.2.3.
- Version 1.1.2 is older than version 1.1.2.4.
- Version 1.2.5 is newer than version 1.2.3.4.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

CompareTo(Object)

Compares the current Version object to a specified object and returns an indication of their relative values.

C#

```
public int CompareTo (object? version);
```

Parameters

`version` Object

An object to compare, or `null`.

Returns

Int32

A signed integer that indicates the relative values of the two objects, as shown in the following table.

Expand table

Return value	Meaning
Less than zero	The current Version object is a version before `version`.
Zero	The current Version object is the same version as `version`.
Greater than zero	The current Version object is a version subsequent to `version`, or `version` is `null`.

Implements

CompareTo(Object)

Exceptions

ArgumentException

`version` is not of type Version.

Remarks

The components of `Version` in decreasing order of importance are: major, minor, build, and revision. An unknown component is assumed to be older than any known component.

For example:

- Version 1.1 is older than version 1.1.0
- Version 1.1 is older than version 1.1.1
- Version 1.1 is older than version 1.1.2.3
- Version 1.1.2 is older than version 1.1.2.4
- Version 1.2.5 is newer than version 1.2.3.4

See also

- CompareTo(Version)
- Equals

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.Equals Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a value indicating whether two Version object represent the same value.

Overloads

Expand table

Equals(Object)	Returns a value indicating whether the current Version object is equal to a specified object.
Equals(Version)	Returns a value indicating whether the current Version object and a specified Version object represent the same value.

Equals(Object)

Returns a value indicating whether the current Version object is equal to a specified object.

C#

```
public override bool Equals (object? obj);
```

Parameters

obj Object

An object to compare with the current Version object, or `null`.

Returns

Boolean

`true` if the current Version object and `obj` are both Version objects, and every component of the current Version object matches the corresponding component of `obj`; otherwise, `false`.

See also

- CompareTo(Object)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Equals(Version)

Returns a value indicating whether the current Version object and a specified Version object represent the same value.

C#

```
public bool Equals (Version? obj);
```

Parameters

`obj` Version

A Version object to compare to the current Version object, or `null`.

Returns

Boolean

`true` if every component of the current Version object matches the corresponding component of the `obj` parameter; otherwise, `false`.

Implements

Equals(T)

Remarks

This method implements the IEquatable<T> interface, and performs slightly better than the Equals method because it does not have to unbox the `obj` parameter.

See also

- Equals(Object)
- CompareTo

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

Version.GetHashCode Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Returns a hash code for the current Version object.

C#

```
public override int GetHashCode ();
```

Returns

Int32

A 32-bit signed integer hash code.

Examples

The following code example demonstrates the Version constructor, and Major, Minor, Build, Revision, MajorRevision, and MinorRevision properties.

C#

```
// This example demonstrates the Version.Revision,
// MajorRevision, and MinorRevision properties.
using System;

class Sample
{
    public static void Main()
    {

    string fmtStd = "Standard version:\n" +
                    "  major.minor.build.revision = {0}.{1}.{2}.{3}";
    string fmtInt = "Interim version:\n" +
                    "  major.minor.build.majRev/minRev = {0}.{1}.{2}.
{3}/{4}";

    Version std = new Version(2, 4, 1128, 2);
    Version interim = new Version(2, 4, 1128, (100 << 16) + 2);
```

```
    Console.WriteLine(fmtStd, std.Major, std.Minor, std.Build,
std.Revision);
    Console.WriteLine(fmtInt, interim.Major, interim.Minor, interim.Build,
                              interim.MajorRevision, interim.MinorRevision);
    }
}
/*
This code example produces the following results:

Standard version:
  major.minor.build.revision = 2.4.1128.2
Interim version:
  major.minor.build.majRev/minRev = 2.4.1128.100/2

*/
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.Parse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

Parse(ReadOnlySpan<Char>)	Converts the specified read-only span of characters that represents a version number to an equivalent Version object.
Parse(String)	Converts the string representation of a version number to an equivalent Version object.

Parse(ReadOnlySpan<Char>)

Converts the specified read-only span of characters that represents a version number to an equivalent Version object.

```
C#

public static Version Parse (ReadOnlySpan<char> input);
```

Parameters

`input` ReadOnlySpan<Char>

A read-only span of characters that contains a version number to convert.

Returns

Version

An object that is equivalent to the version number specified in the `input` parameter.

Exceptions

ArgumentException

`input` has fewer than two or more than four version components.

ArgumentOutOfRangeException

At least one component in `input` is less than zero.

FormatException

At least one component in `input` is not an integer.

OverflowException

At least one component in `input` represents a number that is greater than Int32.MaxValue.

Remarks

The `input` parameter must have the following format:

```
major.minor[.build[.revision]]
```

where `major`, `minor`, `build`, and `revision` are the string representations of the version number's four components: major version number, minor version number, build number, and revision number, respectively. Optional components are shown in square brackets ([and]). The components must appear in the specified order and must be separated by periods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Parse(String)

Converts the string representation of a version number to an equivalent Version object.

```
C#
```

```
public static Version Parse (string input);
```

Parameters

`input` String

A string that contains a version number to convert.

Returns

Version

An object that is equivalent to the version number specified in the `input` parameter.

Exceptions

ArgumentNullException

`input` is `null`.

ArgumentException

`input` has fewer than two or more than four version components.

ArgumentOutOfRangeException

At least one component in `input` is less than zero.

FormatException

At least one component in `input` is not an integer.

OverflowException

At least one component in `input` represents a number that is greater than Int32.MaxValue.

Examples

The following example uses the Parse method to parse a number of strings that contain version information.

```
C#

using System;

public class Example
```

```
{
   public static void Main()
   {
      string input = "4.0";
      ParseVersion(input);

      input = "4.0.";
      ParseVersion(input);

      input = "1.1.2";
      ParseVersion(input);

      input = "1.1.2.01702";
      ParseVersion(input);

      input = "1.1.2.0702.119";
      ParseVersion(input);

      input =  "1.3.5.2150000000";
      ParseVersion(input);
   }

   private static void ParseVersion(string input)
   {
      try {
         Version ver = Version.Parse(input);
         Console.WriteLine("Converted '{0} to {1}.", input, ver);
      }
      catch (ArgumentNullException) {
         Console.WriteLine("Error: String to be parsed is null.");
      }
      catch (ArgumentOutOfRangeException) {
         Console.WriteLine("Error: Negative value in '{0}'.", input);
      }
      catch (ArgumentException) {
         Console.WriteLine("Error: Bad number of components in '{0}'.",
                           input);
      }
      catch (FormatException) {
         Console.WriteLine("Error: Non-integer value in '{0}'.", input);
      }
      catch (OverflowException) {
         Console.WriteLine("Error: Number out of range in '{0}'.",
input);
      }
   }
}
// The example displays the following output:
//       Converted '4.0 to 4.0.
//       Error: Non-integer value in '4.0.'.
//       Converted '1.1.2 to 1.1.2.
//       Converted '1.1.2.01702 to 1.1.2.1702.
//       Error: Bad number of components in '1.1.2.0702.119'.
//       Error: Number out of range in '1.3.5.2150000000'.
```

Remarks

The `input` parameter must have the following format:

`major.minor[.build[.revision]]`

where `major`, `minor`, `build`, and `revision` are the string representations of the version number's four components: major version number, minor version number, build number, and revision number, respectively. Optional components are shown in square brackets ([and]). The components must appear in the specified order and must be separated by periods.

> **Important**
>
> Because the string representation of a version number must conform to a recognized pattern, applications should always use exception handling when calling the **Parse** method to parse user input. Alternatively, you can call the **TryParse** method to parse the string representation of a version number and return a value that indicates whether the parse operation succeeded.

The Parse method is a convenience method; it is equivalent to calling the Version(String) constructor.

See also

- ToString
- TryParse(String, Version)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.ToString Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Converts the value of the current Version object to its equivalent String representation.

Overloads

Expand table

ToString()	Converts the value of the current Version object to its equivalent String representation.
ToString(Int32)	Converts the value of the current Version object to its equivalent String representation. A specified count indicates the number of components to return.

ToString()

Converts the value of the current Version object to its equivalent String representation.

C#

```
public override string ToString ();
```

Returns

String

The String representation of the values of the major, minor, build, and revision components of the current Version object, as depicted in the following format. Each component is separated by a period character ('.'). Square brackets ('[' and ']') indicate a component that will not appear in the return value if the component is not defined:

major.minor[.build[.revision]]

For example, if you create a Version object using the constructor `Version(1,1)`, the returned string is "1.1". If you create a Version object using the constructor `Version(1,3,4,2)`, the returned string is "1.3.4.2".

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

ToString(Int32)

Converts the value of the current Version object to its equivalent String representation. A specified count indicates the number of components to return.

C#

```
public string ToString (int fieldCount);
```

Parameters

`fieldCount` Int32

The number of components to return. The `fieldCount` ranges from 0 to 4.

Returns

String

The String representation of the values of the major, minor, build, and revision components of the current Version object, each separated by a period character ('.'). The `fieldCount` parameter determines how many components are returned.

Expand table

fieldCount	Return Value
0	An empty string ("").
1	major
2	major.minor
3	major.minor.build
4	major.minor.build.revision

For example, if you create Version object using the constructor `Version(1,3,5)`, `ToString(2)` returns "1.3" and `ToString(4)` throws an exception.

Exceptions

ArgumentException

`fieldCount` is less than 0, or more than 4.

-or-

`fieldCount` is more than the number of components defined in the current Version object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryFormat(Span<Char>, Int32, Int32)	Tries to format this version instance into a span of characters.
TryFormat(Span<Byte>, Int32, Int32)	Tries to format this version instance into a span of bytes.
TryFormat(Span<Char>, Int32)	Tries to format this version instance into a span of characters.
TryFormat(Span<Byte>, Int32)	Tries to format this version instance into a span of bytes.

TryFormat(Span<Char>, Int32, Int32)

Tries to format this version instance into a span of characters.

C#

```
public bool TryFormat (Span<char> destination, int fieldCount, out int
charsWritten);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`fieldCount` Int32

The number of components to return. This value ranges from 0 to 4.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryFormat(Span<Byte>, Int32, Int32)

Tries to format this version instance into a span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, int fieldCount, out
int bytesWritten);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of UTF-8 bytes.

`fieldCount` Int32

The number of components to return. This value ranges from 0 to 4.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

TryFormat(Span<Char>, Int32)

Tries to format this version instance into a span of characters.

```
C#

public bool TryFormat (Span<char> destination, out int charsWritten);
```

Parameters

`destination` Span<Char>

The span in which to write this instance's value formatted as a span of characters.

`charsWritten` Int32

When this method returns, contains the number of characters that were written in `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

TryFormat(Span<Byte>, Int32)

Tries to format this version instance into a span of bytes.

C#

```
public bool TryFormat (Span<byte> utf8Destination, out int bytesWritten);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of UTF-8 bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.TryParse Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Overloads

Expand table

TryParse(String, Version)	Tries to convert the string representation of a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.
TryParse(ReadOnlySpan<Char>, Version)	Tries to convert the specified read-only span of characters representing a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.

TryParse(String, Version)

Tries to convert the string representation of a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.

C#

```
public static bool TryParse (string? input, out Version? result);
```

Parameters

input String

A string that contains a version number to convert.

result Version

When this method returns, contains the Version equivalent of the number that is contained in `input`, if the conversion succeeded. If `input` is `null`, Empty, or if the conversion fails, `result` is `null` when the method returns.

Returns

Boolean

`true` if the `input` parameter was converted successfully; otherwise, `false`.

Examples

The following example uses the TryParse method to parse a number of strings that contain version information.

C#

```
using System;

public class Example
{
   public static void Main()
   {
      string input = "4.0";
      ParseVersion(input);

      input = "4.0.";
      ParseVersion(input);

      input = "1.1.2";
      ParseVersion(input);

      input = "1.1.2.01702";
      ParseVersion(input);

      input = "1.1.2.0702.119";
      ParseVersion(input);

      input =  "1.3.5.2150000000";
      ParseVersion(input);
   }

   private static void ParseVersion(string input)
   {
      Version ver = null;
      if (Version.TryParse(input, out ver))
         Console.WriteLine("Converted '{0} to {1}.", input, ver);
      else
         Console.WriteLine("Unable to determine the version from '{0}'.",
                           input);
   }
}
// The example displays the following output:
//       Converted '4.0 to 4.0.
//       Unable to determine the version from '4.0.'.
//       Converted '1.1.2 to 1.1.2.
```

```
//       Converted '1.1.2.01702 to 1.1.2.1702.
//       Unable to determine the version from '1.1.2.0702.119'.
//       Unable to determine the version from '1.3.5.2150000000'.
```

Remarks

The `TryParse` method is similar to the Parse method, except that it doesn't throw an exception if the conversion fails. Instead, it returns `false` if `input` is `null`, has fewer than two or more than four components, has at least one component that is not an integer, has at least one component that is less than zero, or has at least one component that is greater than Int32.MaxValue.

For the parse operation to succeed, the `input` parameter must be in the following format:

`major.minor[.build[.revision]]`

where `major`, `minor`, `build`, and `revision` are the string representations of the version number's four components: major version number, minor version number, build number, and revision number, respectively. Optional components are shown in square brackets ([and]). The components must appear in order and must be separated by periods.

See also

- Parse(String)
- ToString

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

TryParse(ReadOnlySpan<Char>, Version)

Tries to convert the specified read-only span of characters representing a version number to an equivalent Version object, and returns a value that indicates whether the conversion succeeded.

```
C#

public static bool TryParse (ReadOnlySpan<char> input, out Version?
result);
```

Parameters

`input` ReadOnlySpan<Char>

A read-only span of characters that contains a version number to convert.

`result` Version

When this method returns, contains the Version equivalent of the number that is contained in `input`, if the conversion succeeded. If `input` is `null`, Empty, or if the conversion fails, `result` is `null` when the method returns.

Returns

Boolean

`true` if the `input` parameter was converted successfully; otherwise, `false`.

Remarks

The `TryParse` method is similar to the Parse method, except that it doesn't throw an exception if the conversion fails. Instead, it returns `false` if `input` is `null`, has fewer than two or more than four components, has at least one component that is not an integer, has at least one component that is less than zero, or has at least one component that is greater than Int32.MaxValue.

For the parse operation to succeed, the `input` parameter must be in the following format:

`major.minor[.build[.revision]]`

where `major`, `minor`, `build`, and `revision` are the string representations of the version number's four components: major version number, minor version number,

build number, and revision number, respectively. Optional components are shown in square brackets ([and]). The components must appear in order and must be separated by periods.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.Equality(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified Version objects are equal.

C#

```
public static bool operator == (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` equals `v2`; otherwise, `false`.

Remarks

The equivalent method for this operator is Version.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.GreaterThan(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the first specified Version object is greater than the second specified Version object.

```
C#
public static bool operator > (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` is greater than `v2`; otherwise, `false`.

Remarks

The equivalent method for this operator is Version.CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.GreaterThanOrEqual(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the first specified Version object is greater than or equal to the second specified Version object.

C#

```
public static bool operator >= (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` is greater than or equal to `v2`; otherwise, `false`.

Remarks

The equivalent method for this operator is Version.CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.Inequality(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether two specified Version objects are not equal.

C#

```
public static bool operator != (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` does not equal `v2`; otherwise, `false`.

Remarks

The equivalent method for this operator is Version.Equals(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.LessThan(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the first specified Version object is less than the second specified Version object.

```
C#
public static bool operator < (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` is less than `v2`; otherwise, `false`.

Exceptions

ArgumentNullException

.NET Framework only: `v1` is `null`.

Remarks

The equivalent method for this operator is Version.CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.LessThanOrEqual(Version, Version) Operator

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Determines whether the first specified Version object is less than or equal to the second Version object.

```
C#
public static bool operator <= (Version? v1, Version? v2);
```

Parameters

v1 Version

The first Version object.

v2 Version

The second Version object.

Returns

Boolean

`true` if `v1` is less than or equal to `v2`; otherwise, `false`.

Exceptions

ArgumentNullException

.NET Framework only: `v1` is `null`.

Remarks

The equivalent method for this operator is Version.CompareTo(Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Equals(Object)
- CompareTo(Object)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.IFormattable.ToString(String, IFormatProvider) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Formats the value of the current instance using the specified format.

C#

```
string IFormattable.ToString (string format, IFormatProvider
formatProvider);
```

Parameters

`format` String

The format to use. -or- A `null` reference (`Nothing` in Visual Basic) to use the default format defined for the type of the IFormattable implementation.

`formatProvider` IFormatProvider

The provider to use to format the value. -or- A `null` reference (`Nothing` in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

String

The value of the current instance in the specified format.

Implements

ToString(String, IFormatProvider)

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Version.ISpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance into the provided span of characters.

C#

```
bool ISpanFormattable.TryFormat (Span<char> destination, out int
charsWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

destination Span<Char>

The span in which to write this instance's value formatted as a span of characters.

charsWritten Int32

When this method returns, contains the number of characters that were written in `destination`.

format ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `destination`.

provider IFormatProvider

An optional object that supplies culture-specific formatting information for `destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	6, 7, 8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Version.IUtf8SpanFormattable.TryFormat Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to format the value of the current instance as UTF-8 into the provided span of bytes.

C#

```
bool IUtf8SpanFormattable.TryFormat (Span<byte> utf8Destination, out int
bytesWritten, ReadOnlySpan<char> format, IFormatProvider provider);
```

Parameters

`utf8Destination` Span<Byte>

The span in which to write this instance's value formatted as a span of bytes.

`bytesWritten` Int32

When this method returns, contains the number of bytes that were written in `utf8Destination`.

`format` ReadOnlySpan<Char>

A span containing the characters that represent a standard or custom format string that defines the acceptable format for `utf8Destination`.

`provider` IFormatProvider

An optional object that supplies culture-specific formatting information for `utf8Destination`.

Returns

Boolean

`true` if the formatting was successful; otherwise, `false`.

Implements

TryFormat(Span<Byte>, Int32, ReadOnlySpan<Char>, IFormatProvider)

Applies to

Product	Versions
.NET	8, 9

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Void Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Specifies a return value type for a method that does not return a value.

```
C#
public struct Void
```

Inheritance Object → ValueType → Void

Remarks

The Void structure is used in the System.Reflection namespace, but is rarely useful in a typical application. The Void structure has no members other than the ones all types inherit from the Object class.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

WeakReference Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.

C#

```
public class WeakReference : System.Runtime.Serialization.ISerializable
```

Inheritance Object → WeakReference

Implements ISerializable

Examples

The following example demonstrates how you can use weak references to maintain a cache of objects as a resource for an application. The cache is constructed using an IDictionary<TKey,TValue> of WeakReference objects keyed by an index value. The Target property for the WeakReference objects is an object in a byte array that represents data.

The example randomly accesses objects in the cache. If an object is reclaimed for garbage collection, a new data object is regenerated; otherwise, the object is available to access because of the weak reference.

C#

```
using System;
using System.Collections.Generic;

public class Program
{
    public static void Main()
    {
        // Create the cache.
        int cacheSize = 50;
        Random r = new Random();
```

```
        Cache c = new Cache(cacheSize);

        string DataName = "";
        GC.Collect(0);

        // Randomly access objects in the cache.
        for (int i = 0; i < c.Count; i++) {
            int index = r.Next(c.Count);

            // Access the object by getting a property value.
            DataName = c[index].Name;
        }
        // Show results.
        double regenPercent = c.RegenerationCount/(double)c.Count;
        Console.WriteLine($"Cache size: {c.Count}, Regenerated:
{regenPercent:P0}");
    }
}

public class Cache
{
    // Dictionary to contain the cache.
    static Dictionary<int, WeakReference> _cache;

    // Track the number of times an object is regenerated.
    int regenCount = 0;

    public Cache(int count)
    {
        _cache = new Dictionary<int, WeakReference>();

        // Add objects with a short weak reference to the cache.
       for (int i = 0; i < count; i++) {
            _cache.Add(i, new WeakReference(new Data(i), false));
        }
    }

    // Number of items in the cache.
    public int Count
    {
        get {  return _cache.Count; }
    }

    // Number of times an object needs to be regenerated.
    public int RegenerationCount
    {
        get { return regenCount; }
    }

    // Retrieve a data object from the cache.
    public Data this[int index]
    {
        get {
            Data d = _cache[index].Target as Data;
            if (d == null) {
```

```
                // If the object was reclaimed, generate a new one.
                Console.WriteLine("Regenerate object at {0}: Yes", index);
                d = new Data(index);
                _cache[index].Target = d;
                regenCount++;
            }
            else {
                // Object was obtained with the weak reference.
                Console.WriteLine("Regenerate object at {0}: No", index);
            }

            return d;
        }
    }
}

// This class creates byte arrays to simulate data.
public class Data
{
    private byte[] _data;
    private string _name;

    public Data(int size)
    {
        _data = new byte[size * 1024];
        _name = size.ToString();
    }

    // Simple property.
    public string Name
    {
        get { return _name; }
    }
}
// Example of the last lines of the output:
//
// ...
// Regenerate object at 36: Yes
// Regenerate object at 8: Yes
// Regenerate object at 21: Yes
// Regenerate object at 4: Yes
// Regenerate object at 38: No
// Regenerate object at 7: Yes
// Regenerate object at 2: Yes
// Regenerate object at 43: Yes
// Regenerate object at 38: No
// Cache size: 50, Regenerated: 94%
```

Remarks

A weak reference allows the garbage collector to collect an object while still allowing an application to access the object. If you need the object, you can still obtain a strong

reference to it and prevent it from being collected. For more information about how to use short and long weak references, see Weak References.

Constructors

Expand table

WeakReference(Object)	Initializes a new instance of the WeakReference class, referencing the specified object.
WeakReference(Object, Boolean)	Initializes a new instance of the WeakReference class, referencing the specified object and using the specified resurrection tracking.
WeakReference(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the WeakReference class, using deserialized data from the specified serialization and stream objects.

Properties

Expand table

IsAlive	Gets an indication whether the object referenced by the current WeakReference object has been garbage collected.
Target	Gets or sets the object (the target) referenced by the current WeakReference object.
Track Resurrection	Gets an indication whether the object referenced by the current WeakReference object is tracked after it is finalized.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Finalize()	Discards the reference to the target represented by the current WeakReference object.

GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo object with all the data needed to serialize the current WeakReference object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GC
- Weak References

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the WeakReference class.

Overloads

Expand table

WeakReference(Object)	Initializes a new instance of the WeakReference class, referencing the specified object.
WeakReference(Object, Boolean)	Initializes a new instance of the WeakReference class, referencing the specified object and using the specified resurrection tracking.
WeakReference(SerializationInfo, StreamingContext)	**Obsolete.** Initializes a new instance of the WeakReference class, using deserialized data from the specified serialization and stream objects.

WeakReference(Object)

Initializes a new instance of the WeakReference class, referencing the specified object.

C#

```
public WeakReference (object? target);
```

Parameters

`target` Object

The object to track or `null`.

Remarks

This constructor creates a short weak reference to `target`.

See also

- Object

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

WeakReference(Object, Boolean)

Initializes a new instance of the WeakReference class, referencing the specified object and using the specified resurrection tracking.

C#

```
public WeakReference (object? target, bool trackResurrection);
```

Parameters

`target` Object

An object to track.

`trackResurrection` Boolean

Indicates when to stop tracking the object. If `true`, the object is tracked after finalization; if `false`, the object is only tracked until finalization.

Examples

The following example creates a cache of data objects with short weak references. This example is part of a larger example provided for the WeakReference class.

```C#
 // Add objects with a short weak reference to the cache.
for (int i = 0; i < count; i++) {
     _cache.Add(i, new WeakReference(new Data(i), false));
 }
```

Remarks

If `trackResurrection` is `false`, a short weak reference is created. If `trackResurrection` is `true`, a long weak reference is created.

See also

- Object
- Boolean
- GCHandle
- Weak References

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

WeakReference(SerializationInfo, StreamingContext)

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Initializes a new instance of the WeakReference class, using deserialized data from the specified serialization and stream objects.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
protected WeakReference (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

An object that holds all the data needed to serialize or deserialize the current WeakReference object.

`context` StreamingContext

(Reserved) Describes the source and destination of the serialized stream specified by `info`.

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

The `context` parameter is reserved, and does not currently participate in this operation.

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference.IsAlive Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an indication whether the object referenced by the current WeakReference object has been garbage collected.

```
C#
public virtual bool IsAlive { get; }
```

Property Value

Boolean

`true` if the object referenced by the current WeakReference object has not been garbage collected and is still accessible; otherwise, `false`.

Remarks

Because an object could potentially be reclaimed for garbage collection immediately after the IsAlive property returns `true`, using this property is not recommended unless you are testing only for a `false` return value.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GC
- Object
- Weak References

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference.Target Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets or sets the object (the target) referenced by the current WeakReference object.

C#

```
public virtual object? Target { get; set; }
```

Property Value

Object

`null` if the object referenced by the current WeakReference object has been garbage collected; otherwise, a reference to the object referenced by the current WeakReference object.

Exceptions

InvalidOperationException

The reference to the target object is invalid. This exception can be thrown while setting this property if the value is a null reference or if the object has been finalized during the set operation.

Examples

The following example tries to obtain an object from a cache of objects with weak references. If the object was reclaimed for garbage collection, a new object is generated. This example is part of a larger example provided for the WeakReference class.

C#

```
Data d = _cache[index].Target as Data;
if (d == null) {
    // If the object was reclaimed, generate a new one.
    Console.WriteLine("Regenerate object at {0}: Yes", index);
```

```
        d = new Data(index);
        _cache[index].Target = d;
        regenCount++;
    }
    else {
        // Object was obtained with the weak reference.
        Console.WriteLine("Regenerate object at {0}: No", index);
    }

    return d;
```

Remarks

After setting this property to the target object, make sure that there are no other strong references to the object; otherwise, it will not be collected.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- Object
- GC
- Weak References

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

WeakReference.TrackResurrection Property

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Gets an indication whether the object referenced by the current WeakReference object is tracked after it is finalized.

C#

```
public virtual bool TrackResurrection { get; }
```

Property Value

Boolean

`true` if the object the current WeakReference object refers to is tracked after finalization; or `false` if the object is only tracked until finalization.

Remarks

If `true`, the weak reference is a long weak reference and `true` was specified for the `trackResurrection` parameter in the WeakReference constructor.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GC
- Weak References

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference.Finalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Discards the reference to the target represented by the current WeakReference object.

C#

```
~WeakReference ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

See also

- GC
- Finalize()

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

WeakReference.GetObjectData(SerializationInfo, StreamingContext) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo object with all the data needed to serialize the current WeakReference object.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization.
It should not be called or extended by application code.",
DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public virtual void GetObjectData
(System.Runtime.Serialization.SerializationInfo info,
System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

An object that holds all the data needed to serialize or deserialize the current WeakReference object.

`context` StreamingContext

(Reserved) The location where serialized data is stored and retrieved.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

Remarks

This method stores all the information in `info` necessary to serialize the current WeakReference object.

The `context` parameter is reserved and is not currently used in the execution of this method.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

See also

- Object
- SerializationInfo

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

WeakReference<T> Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a typed weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.

C#

```
public sealed class WeakReference<T> :
System.Runtime.Serialization.ISerializable where T : class
```

Type Parameters

T

The type of the object referenced.

Inheritance Object → WeakReference<T>

Implements ISerializable

Remarks

A weak reference enables the garbage collector to collect an object while still allowing an application to access the object. If you need the object, you can obtain a strong reference to it and prevent it from being collected.

Constructors

Expand table

WeakReference<T>(T)	Initializes a new instance of the WeakReference<T> class that references the specified object.

WeakReference<T>(T, Boolean)	Initializes a new instance of the WeakReference<T> class that references the specified object and uses the specified resurrection tracking.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Finalize()	Discards the reference to the target that is represented by the current WeakReference<T> object.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetObjectData(SerializationInfo, StreamingContext)	**Obsolete.** Populates a SerializationInfo object with all the data necessary to serialize the current WeakReference<T> object.
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
SetTarget(T)	Sets the target object that is referenced by this WeakReference<T> object.
ToString()	Returns a string that represents the current object. (Inherited from Object)
TryGetTarget(T)	Tries to retrieve the target object that is referenced by the current WeakReference<T> object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

WeakReference<T> Constructors

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Initializes a new instance of the WeakReference<T> class.

Overloads

Expand table

WeakReference<T>(T)	Initializes a new instance of the WeakReference<T> class that references the specified object.
WeakReference<T>(T, Boolean)	Initializes a new instance of the WeakReference<T> class that references the specified object and uses the specified resurrection tracking.

WeakReference<T>(T)

Initializes a new instance of the WeakReference<T> class that references the specified object.

C#

```
public WeakReference (T target);
```

Parameters

`target` T

The object to reference, or `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

WeakReference<T>(T, Boolean)

Initializes a new instance of the WeakReference<T> class that references the specified object and uses the specified resurrection tracking.

C#

```
public WeakReference (T target, bool trackResurrection);
```

Parameters

`target` T

The object to reference, or `null`.

`trackResurrection` Boolean

`true` to track the object after finalization; `false` to track the object only until finalization.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference<T>.Finalize Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Discards the reference to the target that is represented by the current WeakReference<T> object.

```
C#
~WeakReference ();
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference<T>.GetObjectData Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

> ⊗ **Caution**
>
> This API supports obsolete formatter-based serialization. It should not be called or extended by application code.

Populates a SerializationInfo object with all the data necessary to serialize the current WeakReference<T> object.

C#

```
[System.Obsolete("This API supports obsolete formatter-based serialization. It should not be called or extended by application code.", DiagnosticId="SYSLIB0051", UrlFormat="https://aka.ms/dotnet-warnings/{0}")]
public void GetObjectData (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);
```

Parameters

`info` SerializationInfo

An object that holds all the data necessary to serialize or deserialize the current WeakReference<T> object.

`context` StreamingContext

The location where serialized data is stored and retrieved.

Implements

GetObjectData(SerializationInfo, StreamingContext)

Attributes ObsoleteAttribute

Exceptions

ArgumentNullException

`info` is `null`.

Applies to

Product	Versions *(Obsolete)*
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7 *(8, 9)*
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WeakReference<T>.SetTarget(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Sets the target object that is referenced by this WeakReference<T> object.

C#

```
public void SetTarget (T target);
```

Parameters

`target` T

The new target object.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

WeakReference<T>.TryGetTarget(T) Method

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Tries to retrieve the target object that is referenced by the current WeakReference<T> object.

```
C#
public bool TryGetTarget (out T target);
```

Parameters

`target` T

When this method returns, contains the target object, if it is available. This parameter is treated as uninitialized.

Returns

Boolean

`true` if the target was retrieved; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback